



Extreme vSLX Installation and User Guide, 20.2.1

Emulating the ExtremeSwitching
SLX 9540 Device

9036695-00 Rev AA
June 2020



Copyright © 2020 Extreme Networks, Inc. All rights reserved.

Legal Notice

Extreme Networks, Inc. reserves the right to make changes in specifications and other information contained in this document and its website without prior notice. The reader should in all cases consult representatives of Extreme Networks to determine whether any such changes have been made.

The hardware, firmware, software or any specifications described or referred to in this document are subject to change without notice.

Trademarks

Extreme Networks and the Extreme Networks logo are trademarks or registered trademarks of Extreme Networks, Inc. in the United States and/or other countries.

All other names (including any product names) mentioned in this document are the property of their respective owners and may be trademarks or registered trademarks of their respective companies/owners.

For additional information on Extreme Networks trademarks, see: www.extremenetworks.com/company/legal/trademarks

Open Source Declarations

Some software files have been licensed under certain open source or third-party licenses. End-user license agreements and open source declarations can be found at: <https://www.extremenetworks.com/support/policies/open-source-declaration/>



Table of Contents

Preface.....	6
Text Conventions.....	6
Documentation and Training.....	8
Getting Help.....	8
Subscribe to Service Notifications.....	8
Providing Feedback.....	9
About This Document.....	10
What's New in This Document.....	10
Emulated Hardware.....	10
Overview of vSLX.....	11
Introduction to vSLX.....	11
Supported functionality.....	11
Support for multiple SLX-OS versions.....	12
SLX-OS features tested.....	12
Hypervisor support.....	13
vSLX limitations.....	13
vSLX Installation.....	14
Installation overview.....	14
vSLX server requirements.....	15
Download vSLX software.....	16
Download Firmware on vSLX.....	16
Install Linux on an x86 server.....	17
Verifying virtualization support.....	17
Creating additional Linux users.....	17
Network-interface setup.....	18
Configuring a static IP address bridge.....	18
Configuring a dynamic IP address bridge.....	19
Copying the software distribution	20
Installing a vSLX lab on a host	21
Upgrading or downgrading vSLX (host installation).....	22
Container installation (multiple vSLX labs).....	22
Installing the xlx bash scripts.....	23
Creating a container for vSLX.....	24
Installing vSLX in a container.....	24
Uninstalling vSLX completely.....	26
vShell Basics.....	27
Overview of vShell entities.....	27
The vShell environment.....	27
Templates and chassis.....	27

Workstations.....	27
Bridges.....	28
Probes.....	28
Links.....	28
Scaling Considerations.....	28
Resource requirements.....	28
Memory swapping.....	29
Virtual machines (VMs).....	29
SLX 9540 scaling calculations.....	29
Up-and-running example.....	29
Basic vShell Tasks.....	31
Starting vShell.....	31
Displaying vShell Keyboard Shortcuts.....	32
Displaying vShell commands.....	32
Displaying Command Options.....	32
Creating a Template.....	32
Creating a Virtual Device.....	33
Deleting a Virtual Device.....	33
Turning on and Turning off a Virtual Device.....	33
Creating a Virtual Workstation.....	34
Connecting to a Virtual Device.....	34
Disconnecting from a Virtual Device.....	34
Virtual Networks.....	35
Overview of virtual networks.....	35
Links	35
Creating Links Between Ports.....	35
Deleting links between ports.....	36
Creating and linking a bridge	36
Creating and linking a probe	37
Linking across vSLX instances with tunnels.....	37
Linking to a Real Interface.....	38
IP Fabric Emulation.....	39
IP Fabric reference topology.....	39
IP Fabric features tested.....	39
IP control-plane features tested.....	39
IP data-plane features tested.....	40
Layer 2 Exchange Emulation.....	41
L2 Exchange reference topology.....	41
Key capabilities.....	41
L2 Exchange features tested.....	42
VPLS features tested.....	42
VLL features tested.....	42
Known L2 Exchange issues.....	42
Advanced vShell Features.....	43
vShell commands from the Linux shell.....	43
User-developed commands and scripts.....	43
Python requirements for extending vShell	43
Implementation flow for vShell extensions.....	43

Syntax of the cmdparser() function.....	44
Implementing a configuration script	45
Guidelines for isolating issues to vSLX.....	47
General guideline.....	47
Appendix A: vShell Command Reference.....	48
clear debug.....	49
connect.....	50
create bridge.....	51
create chassis.....	53
create link.....	54
create probe.....	56
create template.....	57
create tunnel.....	58
create workstation.....	59
delete.....	60
help.....	62
list.....	63
poweroff.....	64
poweron.....	65
show.....	66
show debug.....	68
start.....	69
system.....	70
vsh.....	71
Appendix B: Alternate Routines.....	72
Multiple vSLX labs (lxc).....	72
Creating a container for vSLX (lxc).....	72
Installing vSLX in a container (lxc).....	75
Appendix C: vSLX on VMware ESXi.....	77
vSLX on VMware overview.....	77
Implementation flow.....	77
Resource Scoping for vSLX on VMware.....	78
vSLX-on-VM scope for OVF export.....	78
Create a VM and install Ubuntu.....	79
Initializing a VM for vSLX.....	79
Specifying VM settings for vSLX.....	79
Installing Ubuntu Linux on the VM.....	80
Installation of vSLX on a VM (linked table).....	80



Preface

This section describes the text conventions used in this document, where you can find additional information, and how you can provide feedback to us.

Text Conventions

Unless otherwise noted, information in this document applies to all supported environments for the products in question. Exceptions, like command keywords associated with a specific software version, are identified in the text.

When a feature, function, or operation pertains to a specific hardware product, the product name is used. When features, functions, and operations are the same across an entire product family, such as ExtremeSwitching switches or SLX routers, the product is referred to as *the switch* or *the router*.

Table 1: Notes and warnings




Icon	Notice type	Alerts you to...
	Tip	Helpful tips and notices for using the product.
	Note	Useful information or instructions.
	Important	Important features or instructions.

Table 1: Notes and warnings (continued)



Icon	Notice type	Alerts you to...
	Caution	Risk of personal injury, system damage, or loss of data.
	Warning	Risk of severe personal injury.

Table 2: Text

Convention	Description
<code>screen displays</code>	This typeface indicates command syntax, or represents information as it appears on the screen.
The words <i>enter</i> and <i>type</i>	When you see the word <i>enter</i> in this guide, you must type something, and then press the Return or Enter key. Do not press the Return or Enter key when an instruction simply says <i>type</i> .
Key names	Key names are written in boldface, for example Ctrl or Esc . If you must press two or more keys simultaneously, the key names are linked with a plus sign (+). Example: Press Ctrl+Alt+Del
<i>Words in italicized type</i>	Italics emphasize a point or denote new terms at the place where they are defined in the text. Italics are also used when referring to publication titles.
NEW!	New information. In a PDF, this is searchable text.

Table 3: Command syntax

Convention	Description
bold text	Bold text indicates command names, keywords, and command options.
<i>italic</i> text	Italic text indicates variable content.
[]	Syntax components displayed within square brackets are optional. Default responses to system prompts are enclosed in square brackets.
{ x y z }	A choice of required parameters is enclosed in curly brackets separated by vertical bars. You must select one of the options.
x y	A vertical bar separates mutually exclusive elements.
< >	Nonprinting characters, such as passwords, are enclosed in angle brackets.
...	Repeat the previous element, for example, <i>member</i> [<i>member</i> . . .].
\	In command examples, the backslash indicates a “soft” line break. When a backslash separates two lines of a command input, enter the entire command at the prompt without the backslash.

Documentation and Training

Find Extreme Networks product information at the following locations:

[Current Product Documentation](#)

[Release Notes](#)

[Hardware and software compatibility](#) for Extreme Networks products

[Extreme Optics Compatibility](#)

[Other resources](#) such as white papers, data sheets, and case studies

Extreme Networks offers product training courses, both online and in person, as well as specialized certifications. For details, visit www.extremenetworks.com/education/.

Getting Help

If you require assistance, contact Extreme Networks using one of the following methods:

Extreme Portal

Search the GTAC (Global Technical Assistance Center) knowledge base; manage support cases and service contracts; download software; and obtain product licensing, training, and certifications.

The Hub

A forum for Extreme Networks customers to connect with one another, answer questions, and share ideas and feedback. This community is monitored by Extreme Networks employees, but is not intended to replace specific guidance from GTAC.

Call GTAC

For immediate support: (800) 998 2408 (toll-free in U.S. and Canada) or 1 (408) 579 2826. For the support phone number in your country, visit: www.extremenetworks.com/support/contact

Before contacting Extreme Networks for technical support, have the following information ready:

- Your Extreme Networks service contract number, or serial numbers for all involved Extreme Networks products
- A description of the failure
- A description of any actions already taken to resolve the problem
- A description of your network environment (such as layout, cable type, other relevant environmental information)
- Network load at the time of trouble (if known)
- The device history (for example, if you have returned the device before, or if this is a recurring problem)
- Any related RMA (Return Material Authorization) numbers

Subscribe to Service Notifications

You can subscribe to email notifications for product and software release announcements, Vulnerability Notices, and Service Notifications.

1. Go to www.extremenetworks.com/support/service-notification-form.
2. Complete the form (all fields are required).

3. Select the products for which you would like to receive notifications.

**Note**

You can modify your product selections or unsubscribe at any time.

4. Select **Submit**.

Providing Feedback

The Information Development team at Extreme Networks has made every effort to ensure the accuracy and completeness of this document. We are always striving to improve our documentation and help you work better, so we want to hear from you. We welcome all feedback, but we especially want to know about:

- Content errors, or confusing or conflicting information.
- Improvements that would help you find relevant information in the document.
- Broken links or usability issues.

If you would like to provide feedback, you can do so in three ways:

- In a web browser, select the feedback icon and complete the online feedback form.
- Access the feedback form at <https://www.extremenetworks.com/documentation-feedback/>.
- Email us at documentation@extremenetworks.com.

Provide the publication title, part number, and as much detail as possible, including the topic heading and page number if applicable, as well as your suggestions for improvement.



About This Document

[What's New in This Document](#) on page 10
[Emulated Hardware](#) on page 10

What's New in This Document

This guide accompanies the SLX-OS 20.2.1 software release. There are no changes to the guide from the previous release.

For more information about the release, see the *Extreme SLX-OS 20.2.1 Release Notes*.

Emulated Hardware

This release supports emulation of the ExtremeSwitching SLX 9540 device.

To obtain information about other releases, refer to the documentation specific to that release.



Overview of vSLX

[Introduction to vSLX on page 11](#)

[Supported functionality on page 11](#)

Introduction to vSLX

Extreme Virtual SLX (vSLX) is a virtual lab that enables you to emulate the supported devices. You can also create virtual networks of workstations, SLX devices, tunnels, bridges, and probes.

You can use vSLX for training, configuration buildout and validation, workflow and automation development, and testing. For example:

- Hands-on training of SLX-OS for CLI and programmatic API of the supported devices
- Building and validation of configuration before applying it to a supported device
- Development and testing of automation scripts and software, independent of hardware
- Development and testing Extreme Workflow Composer (EWC) workflows
- Configuration of management plane functionality (CLI and programmatic API) for SLX-OS features

There are two supported installation contexts:

- Host installation: All users share one virtual lab.
- Installation in Linux containers: Multiple users have independent virtual labs.



Note

Although using vSLX does not require Linux experience, installing vSLX does require basic Linux experience.

Supported functionality

Under vSLX, you can emulate most functionality of the supported devices:

- CLI configurations
- Most routing and switching control-plane functions
- All physical ports

Support for multiple SLX-OS versions

Virtual labs can include devices running different SLX-OS versions, as follows:

Table 4: Version-support matrix

vSLX version	SLX-OS versions supported
vSLX 2.3.0	20.1.1 18r.2.00 18r.1.01 18r.1.00
vSLX 2.1.0	18r.2.00 18r.1.01 18r.1.00
vSLX 2.0.0	18r.1.01 18r.1.00

For support of multiple SLX-OS versions, refer to the "Upgrade to the new vSLX version—retaining configurations" table in [Installation overview](#) on page 14.

SLX-OS features tested

The following infrastructure and control-plane features have been tested for this release:

- Basic infrastructure
- BFD
- BGP and BGP IPv6
- ISIS and ISIS IPv6
- L3VPN
- LAG (port-channels)
- LDP
- MCT
- MPLS-TE
- OSPF and OSPFV3
- Service and Monitoring (syslog, SNMPWALK, and SNMP traps)
- STP, RSTP, and MSTP
- VLAN
- VRRP and VRRPv3

For IP Fabric features tested for this release, refer to [IP Fabric features tested](#) on page 39.

For L2 Exchange features tested for this release, refer to [L2 Exchange features tested](#) on page 42

Hypervisor support

The KVM/QEMU hypervisor is supported, and is the preferred method for deployment.

For details of vSLX VMware (ESXi) implementation, refer to [Appendix C: vSLX on VMware ESXi](#) on page 77.

vSLX limitations

vSLX limitations are as follows:

- Although L2/L3 and single-VTEP VXLAN data planes are supported, data-plane emulation does not support heavy traffic for data-plane service deployment. The data plane is supported only for verification of control-plane deployment using ping, traceroute, and so forth.
- Cloud deployment of vSLX has not been tested.
- vSLX supports network packet broker (NPB) system-mode only for exercising the command and control portion of the network. Therefore, solution support is limited to Extreme Visibility manager and leveraging or verification of SSH, SNMP, Syslog, and OpenFlow. Integration of vSLX and SessionDirector is not currently emulated.



vSLX Installation

- [Installation overview on page 14](#)
- [Download vSLX software on page 16](#)
- [Download Firmware on vSLX on page 16](#)
- [Install Linux on an x86 server on page 17](#)
- [Network-interface setup on page 18](#)
- [Copying the software distribution on page 20](#)
- [Installing a vSLX lab on a host on page 21](#)
- [Container installation \(multiple vSLX labs\) on page 22](#)
- [Uninstalling vSLX completely on page 26](#)

Installation overview

The following tables give overviews of the clean and the upgrade installation flows.

Table 5: Clean installation of the new vSLX version

Task	Topics
Download software	Download vSLX software on page 16
Install Ubuntu Linux	Install Linux on an x86 server on page 17
Create an IP-address bridge	Perform one of the following tasks: <ul style="list-style-type: none">• Configuring a static IP address bridge on page 18• Configuring a dynamic IP address bridge on page 19

Table 5: Clean installation of the new vSLX version (continued)

Task	Topics
Copy downloaded software	Copying the software distribution on page 20
Install vSLX	Perform one of the following tasks: <ul style="list-style-type: none"> • Installing a vSLX lab on a host on page 21 • Container installation (multiple vSLX labs) on page 22

Table 6: Upgrade to the new vSLX version—retaining configurations

Task	Host installation	Container installation
Download software	Download vSLX software on page 16	Download vSLX software on page 16
Copy downloaded software	Copying the software distribution on page 20	Copying the software distribution on page 20
Delete and restore x1xc bash scripts	—	Reinstalling the x1xc bash scripts on page 23
Uninstall the previous vSLX version and install the new version.	Upgrading or downgrading vSLX (host installation) on page 22	Reinstalling vSLX in a container on page 25

vSLX server requirements

vSLX is hosted on an x86 server platform; server requirements depend on the scale of the deployment.

At the low end, a laptop with 12 GB of memory can be used to emulate a single virtual device. The following table lists the CPU, memory, and storage requirements of a vSLX server.

Table 7: vSLX server requirements

Scope	CPUs	Memory required	Storage required
One device	i5 (4 cores)	12 GB	40 GB (SLX 9540)
Network	E5 (or better) Minimum: 12 cores Recommended: 24–32 or more cores	Minimum: 64 GB Recommended: 128 GB	1 TB

For scoping details, refer to [Scaling Considerations](#) on page 28.

Download vSLX software

Download the required software components to a folder named `vs1x-re1` on your workstation.



Note

If you have basic support entitlement, go to <http://support.extremenetworks.com>, search for "vSLX", and choose the appropriate vSLX software option.

Table 8: Files required to download from Extreme

File	Notes
<code>vs1x-x.x.x.tar.gz</code>	Contains the vSLX emulator distribution.
<code>slxos-version.x.xx.tar.gz</code>	Contains the Extreme SLX-OS for the virtual device application distribution.

Download Firmware on vSLX

Firmware download is supported on virtual devices over Linux. It is not supported on SLX-OS CLI.

1. Log in to SLX-OS as admin user.
2. Switch to Linux shell using the **start-shell** command.

```
SLX# start-shell
```

3. Get the Linux admin user shell using the **su** command. Enter the default root password `fibranne` when prompted.

```
[admin@SLX]# su
Password:
```

4. Download the firmware using options such as `-p` (FTP or SCP), `-j` (perform vSLX-OS upgrade).

```
Help :
firmwaredownload -h

firmwaredownload syntax:
firmwaredownload -sjb -S -p <protocol> <serverip>, <username>, <image full path>,
<password>

firmware download using FTP:
firmwaredownload -sjb -S -p ftp 10.10.10.10,<releaseuser>,<image full
path>,<releaseuser>

firmware download using SCP:
firmwaredownload -sjb -S -p scp 10.10.10.10,<releaseuser>,<image full
path>,<releaseuser>
```



Note

- There is no FWDL support across releases for SLX 9540 vSLX.
- There is no **fullinstall** support.

5. After downloading the firmware, verify the firmware version using SLX-OS CLI command, **show version**.

```
SLX# show version
```


Install Linux on an x86 server

vSLX runs on Ubuntu Linux, installed on an x86 server.

Note the following installation guidelines:

- The only supported Ubuntu version is Ubuntu Server 16.04.x LTS, which you can download from <https://www.ubuntu.com/>.
- You can either install Ubuntu as a clean installation or upgrade from a previous version.
- Include the OpenSSH server package in your installation.
- For detailed instructions, refer to the Ubuntu website.

The top-level steps are as follows.

1. Create a bootable USB stick by using the Ubuntu ISO image. Click the following link for detailed instructions: <https://www.ubuntu.com/download/desktop/create-a-usb-stick-on-ubuntu>.
2. Boot your server from the USB stick and follow the instructions to install the Ubuntu distribution on the hard disk.
3. Log in with the user and password you created during installation, and verify that you have `sudo` privileges.

Verifying virtualization support

You need to verify that your x86 computer is configured to support hardware virtualization.



Note

The syntax used in this task is valid for Intel x86 CPUs. For other supported CPUs, refer to the appropriate resources.

1. Log in to Linux, and enter the following command.

```
user@ubuntu:~$ cat /proc/cpuinfo | grep vmx
```

If the output contains `vmx` flags, your system is already configured for hardware virtualization.

2. If the output does not contain `vmx` flags:
 - a. Reboot the computer and open the BIOS menu.
 - b. Enable hardware virtualization, save, and exit.
 - c. Log in to Linux, and enter the following command to verify that virtualization is now enabled.

```
user@ubuntu:~$ cat /proc/cpuinfo | grep vmx
```

If the output contains `vmx` flags, your system is now configured for hardware virtualization.

Creating additional Linux users

If needed, create additional users with Linux `sudo` privileges on the x86 server.

If you will be implementing [Container installation \(multiple vSLX labs\)](#) on page 22, best practice is to create a user for each container-based virtual lab.

1. Create a user.

```
user@ubuntu:~$ sudo adduser --shell /bin/bash --home /home/vlab1_user vlab1_user
```

- When prompted, set the password.

```
Enter new UNIX password:
*****
```

- Add the user to the `sudo` group.

```
user@ubuntu:~$ sudo usermod -a -G sudo vlab1_user
```

- To verify that the new user is created and belongs to the `sudo` group, enter the **`sudo groups`** command.

```
user@ubuntu:~$ sudo groups vlab1_user
```

Network-interface setup

You need to create a bridge to connect the server external management interface and the vSLX management interface.

The bridge implementation varies if Dynamic Host Configuration Protocol (DHCP) is employed or not:

- If you are not using DHCP, perform [Configuring a static IP address bridge](#) on page 18.
- If you are using DHCP, perform [Configuring a dynamic IP address bridge](#) on page 19.

Configuring a static IP address bridge

If your network configuration is not controlled by a DHCP server, perform this task to configure a static IP address bridge.

- (If required) To retrieve the IP address and the netmask, enter **`ifconfig`** *interface_name*.
- Enter **`ifconfig -a`** to verify that the Ethernet interface is up.

```
user@ubuntu:~$ ifconfig -a
```

- Enter the **`ping`** command—specifying a known URL—to verify that you have internet connectivity.

```
user@ubuntu:~$ ping example.com
```

- Install `bridge-utils`, using the following commands:

```
user@ubuntu:~$ sudo apt-get update
user@ubuntu:~$ sudo apt-get install bridge-utils
```

- Enter **`sudo vi /etc/network/interfaces`** to modify the following interfaces example with the values you retrieved, replace "eth0" with the actual server external interface, and replace all other `< >` with your values.

```
auto lo
iface lo inet loopback

auto eth0
iface eth0 inet static
address 0.0.0.0

auto br0
iface br0 inet static
address <ip-address>
netmask <net-mask>
gateway <gateway-ip-address>
dns-nameserver <dns-nameservers>
dns-search <search-domains>
bridge_ports <eth0>
bridge_stp off
```

```
bridge_fd 0
bridge_maxwait 0
```

6. Save and close the file.
7. Reboot the server.

```
user@ubuntu:~$ sudo reboot
```

Configuring a dynamic IP address bridge

If your network configuration is controlled by a DHCP server, perform this task to configure a dynamic IP address bridge.

1. To display the available network interfaces, enter **cat /etc/network/interfaces**.

```
user @ubuntu:~$ cat /etc/network/interfaces

# This file describes the network interfaces available on your system
# and how to activate them. For more information, see interfaces(5).

source /etc/network/interfaces.d/*

# The loopback network interface
auto lo
iface lo inet loopback

# The primary network interface
auto ens3
iface ens3 inet dhcp
```

2. Enter **ifconfig -a** to verify that the Ethernet interface is up.

```
user@ubuntu:~$ ifconfig -a
```

3. Enter the **ping** command—specifying a known URL—to verify that you have internet connectivity.

```
user@ubuntu:~$ ping example.com
```

4. Install **bridge-utils**, using the following commands:

```
user@ubuntu:~$ sudo apt-get update
user@ubuntu:~$ sudo apt-get install bridge-utils
```

5. Enter **sudo vi /etc/network/interface** to modify the interfaces file.

```
user@ubuntu:~$ sudo vi /etc/network/interface
```

6. Define the **br0** bridge and enslave the primary network interface (**ens** in the following example) under **br0**.

```
# This file describes the network interfaces available on your system
# and how to activate them. For more information, see interfaces(5).

source /etc/network/interfaces.d/*

# The loopback network interface
auto lo
iface lo inet loopback

# The primary network interface
# We commented out the following two lines:
#auto ens3
#iface ens3 inet dhcp

# We added the following definition of br0:
auto br0
```

```
iface br0 inet dhcp
    bridge_ports ens3
    bridge_stp off
    bridge_fd 0
    bridge_maxwait 0
```

7. Save and close the file.
8. Reboot the server.

```
user@ubuntu:~$ sudo reboot
```

9. After logging back in, enter **ifconfig -a** to verify that the IP address is now assigned to br0.

```
user@ubuntu:~$ ifconfig -a
br0      Link encap:Ethernet  HWaddr 52:55:00:d1:55:01
         inet addr:10.0.0.110  Bcast:10.0.0.255  Mask:255.255.255.0
         inet6 addr: 2601:644:8780:2e56:e99d:2411:3c93:b7e6/64 Scope:Global
         inet6 addr: 2601:644:8780:2e56:5055:ff:fed1:5501/64 Scope:Global
         inet6 addr: fe80::5055:ff:fed1:5501/64 Scope:Link
         UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
         RX packets:142 errors:0 dropped:0 overruns:0 frame:0
         TX packets:93 errors:0 dropped:0 overruns:0 carrier:0
         collisions:0 txqueuelen:1000
         RX bytes:18815 (18.8 KB)  TX bytes:13887 (13.8 KB)

ens3     Link encap:Ethernet  HWaddr 52:55:00:d1:55:01
         inet6 addr: fe80::5055:ff:fed1:5501/64 Scope:Link
         UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
         RX packets:142 errors:0 dropped:0 overruns:0 frame:0
         TX packets:103 errors:0 dropped:0 overruns:0 carrier:0
         collisions:0 txqueuelen:1000
         RX bytes:20803 (20.8 KB)  TX bytes:14793 (14.7 KB)

lo       Link encap:Local Loopback
         inet addr:127.0.0.1  Mask:255.0.0.0
         inet6 addr: ::1/128 Scope:Host
         UP LOOPBACK RUNNING  MTU:65536  Metric:1
         RX packets:80 errors:0 dropped:0 overruns:0 frame:0
         TX packets:80 errors:0 dropped:0 overruns:0 carrier:0
         collisions:0 txqueuelen:1
         RX bytes:5920 (5.9 KB)  TX bytes:5920 (5.9 KB)
```

Copying the software distribution

Copy the downloaded files from your workstation to the Linux server on which you are installing vSLX, and list them.

1. On the Linux server, create an SLX distribution directory and change to it.

```
user@ubuntu:~$ sudo mkdir /slxos-dist
user@ubuntu:~$ cd /slxos-dist
```



Note

For multiple SLX-OS versions, use a separate directory for each version of SLX-OS.

- Use the **scp** command to copy the compressed distribution files from your workstation to the `slxos-dist` directory on the Linux server; then list them.



Note

The file names in this flow are for release 17r.2.00. Substitute the names of the actual files you downloaded in [Download vSLX software](#) on page 16.

```
user@ubuntu:/slxos-dist$ sudo scp user@192.0.2.0:/home/user/work/vslx-rel/
vslx2.0.0.tar.gz .
user@ubuntu:/slxos-dist$ sudo scp user@192.0.2.0:/home/user/work/vslx-rel/slxos17r.
2.00.tar.gz .
user@ubuntu:/slxos-dist$ ls
slxos17r.2.00.tar.gz  vslx2.0.0.tar.gz
```

- Unpack the SLX-OS distribution and then display the contents of the `slxos-dist` directory.

```
user@ubuntu:/slxos-dist$ sudo tar -zxf slxos17r.2.00.tar.gz
user@ubuntu:/slxos-dist$ ls
slxos17r.2.00  slxos17r.2.00.tar.gz  vslx2.0.0.tar.gz
```

- Display the contents of the release directory (for example, `slxos17r.2.00`).

```
user@ubuntu:/slxos-dist$ ls slxos17r.2.00
app_names                onie-host-installer      SWBD2001
breeze_tools             onie-installer           SWBD2002
breeze_tools.tgz         onie-installer-files.tgz SWBD2003
Brocade_SLX_CatenatedMIBs.tar.gz onie-target              SWBD2016
brocade_slx_proto_models.tar.gz oss-binaries-x86_64-ubuntu-14.04-rootfs.tar.gz
SWBD2017
common                   platform_names           SWBD2500
install                  pubkey.pem               SWBD2900
install_cr               signature2.tar           SWBD4000
install_pbr              signature2.tar.sig       vnos-deinstall
install_verify           signature.tar            vnos-install
libbrcmsdk.a.SWBD2000.tgz signature.tar.sig        vslx-utils_1.0.0.deb
libbrcmsdk.a.SWBD2016.tgz slx17r200_yang.tar.gz
libbrcmsdk.a.SWBD4000.tgz SWBD2000
```

- Unpack the vSLX distribution and then display the contents of the `slxos-dist` directory.

```
user@ubuntu:/slxos-dist$ sudo tar -zxf vslx2.0.0.tar.gz
user@ubuntu:/slxos-dist$ ls
extreme-lxc_2.0.0.deb  slxos17r.2.00  slxos17r.2.00.tar.gz  vslx2.0.0.tar.gz
vslx_2.0.0.deb
```

Installing a vSLX lab on a host

Perform this task to install a shared virtual lab on an x86 server.



Note

If you require independent virtual labs on a server, refer to [Container installation \(multiple vSLX labs\)](#) on page 22.

The vSLX host distribution package contains the software components required to virtualize supported SLX devices on a x86 server.

- Enter the **dpkg** command to install the vSLX Debian package on the host.

```
user@ubuntu:~$ sudo dpkg -i /slxos-dist/vslx_2.0.0.deb
```

- Run the **postinst-setup.sh** utility to install the dependent packages and configure system files.

```
user@ubuntu:~$ /VM/postinst-setup.sh
```

After successful installation, refer to [Up-and-running example](#) on page 29.

Upgrading or downgrading vSLX (host installation)

Use this task to uninstall a vSLX host-installation—preserving your vSLX objects and configurations—and reinstall a different vSLX version.



Note

This flow installs new vSLX and SLX-OS versions—while maintaining support for existing and new objects of currently installed SLX-OS versions. But if you need a clean upgrade, refer to [Uninstalling vSLX completely](#) on page 26.

1. In vShell, make sure that all devices (chassis and workstations) are powered off.

```
(vsh) poweroff ch_9540_01
```

2. Make sure that any host VMs are closed.
3. Uninstall the current vSLX distribution—preserving your vSLX objects and configurations.

```
user@ubuntu:~$ sudo dpkg -r vslx
```

4. Enter the **dpkg -i** command, specifying the new vSLX package.

```
user@ubuntu:~$ sudo dpkg -i vslx_x.x.x.deb
```

5. Enter the **postinst-setup.sh** command, specifying the new post-installation package.

```
user@ubuntu:~$ sudo /VM/postinst-setup.sh
```

6. Reboot the server.

```
user@ubuntu:~$ sudo reboot
```

Container installation (multiple vSLX labs)

Installing vSLX in Linux containers—rather than directly on a host—enables each user to have an independent virtual lab.

Linux Containers (LXC) support running multiple Linux systems on a control host using a single Linux kernel. We use *privileged* containers, created by root and running as root. After creating containers, you install one vSLX virtual lab in each container.

The recommended implementation flow utilizes a Linux bash script that we have developed. (This script enables **xlxc** commands, based on the Linux **lxc** commands.):

- (Once) [Installing the xlxc bash scripts](#) on page 23
- (For each lab) [Creating a container for vSLX](#) on page 24
- (For each lab) [Installing vSLX in a container](#) on page 24

The advantages of the **xlxc** flow over directly using **lxc** are as follows:

- There is no need to edit the default Linux container-configuration file.
- The command syntax is less complex.

- There are fewer steps.



Note

If you need container installation of vSLX using **lxc** commands (and manual editing of the container-configuration file), refer to [Multiple vSLX labs \(lxc\)](#) on page 72.

Installing the xlxc bash scripts

This task is a prerequisite for the recommended flow of installing and managing vSLX containers.



Note

You need to perform this task only once per physical or virtual device.

1. Log in to the x86 server as a user with sudo privileges.



Note

Although all sudo users can create and access all containers, our user in this flow is `vlabl_user`, created in [Creating additional Linux users](#) on page 17.

2. Install **lxc** using the following commands, confirming prompts to continue.

```
vlabl_user@ubuntu:~$ sudo apt-get update
vlabl_user@ubuntu:~$ sudo apt-get install lxc
```

3. Install the **xlxc** bash scripts, which are wrappers over the Linux **lxc** commands.

```
vlabl_user@ubuntu:~$ sudo dpkg -i /slxos-dist/extreme-lxc_2.0.0.deb
```

4. For help on the **xlxc** options:

- For a list of sub-commands, enter **xlxc help**.

```
vlabl_user@ubuntu:~$ xlxc help
```

- For help on a sub-command, enter **xlxc help <sub-command>**.

```
vlabl_user@ubuntu:~$ xlxc help create
```

Reinstalling the xlxc bash scripts

This task is a prerequisite for upgrading or downgrading vSLX in Linux containers.



Note

You need to perform this task only once per physical or virtual device.

1. In vShell, make sure that all devices (chassis and workstations) are powered off.

```
(vsh) poweroff ch_9540_01
```

2. Make sure that any container VMs are closed.
3. Log in to the x86 server as a user with sudo privileges.
4. Uninstall the current **xlxc** bash scripts.

```
vlabl_user@ubuntu:~$ sudo dpkg -r extreme-lxc
```

5. Install the replacement **xlxc** bash scripts.

```
vlabl_user@ubuntu:~$ sudo dpkg -i /slxos-dist/extreme-lxc_2.1.0.deb
```

Creating a container for vSLX

(For the **xlxc** flow only) This task is a prerequisite for installing each container instance of vSLX.

1. Make sure that you are logged in to the x86 server as a user with sudo privileges.
2. Enter the **xlxc create** command to create the container.

```

vlab1_user@ubuntu:~$ xlxc create VLAB1
Using image from local cache
Unpacking the rootfs
---
You just created an Ubuntu xenial amd64 (20180701_07:42) container.
To enable SSH, run: apt install openssh-server
No default root or user password are set by LXC.
Container config is patched
Container root account is enabled with no password
Container VLAB1 is created successfully

```

3. Enter the **xlxc mount** command.

```

vlab1_user@ubuntu:~$ xlxc mount VLAB1 /slxos-dist /slxos-dist
Host '/slxos-dist' is mounted on '/slxos-dist' in container 'VLAB1'

```

Installing vSLX in a container

(For the **xlxc** flow only) After preparing a Linux container, perform this task to install a vSLX virtual lab in the container.

1. Start the container and log in as root (with no password required).

```

vlab1_user@ubuntu:~$ xlxc start VLAB1
Connected to tty 0
Type <Ctrl+a q> to exit the console, <Ctrl+a Ctrl+a> to enter Ctrl+a itself
(output
                truncated)
Ubuntu 16.04.4 LTS VLAB1 console
VLAB1 login:

```

2. Verify that all is ready for vSLX installation, as follows:

- a. List the `slxos-dist` directory.

```

root@VLAB1:~# ls /slxos-dist
extreme-lxc_2.0.0.deb  slxos17r.2.00  slxos17r.2.00.tar.gz  vslx_2.0.0.deb
vslx2.0.0.tar.gz

```



Note

You can also list included directories.

- b. Verify that the container's Ethernet interface is up and that an internet address is assigned.

```

root@VLAB1:~# ifconfig -a
eth0      Link encap:Ethernet  HWaddr 00:16:3e:b5:7d:b9
          inet addr:192.0.2.0  Bcast:10.0.0.255  Mask:255.255.255.0
          inet6 addr: 2001:db8::/64 Scope:Link
          inet6 addr: 2001:db8:ffff:ffff:ffff:ffff:ffff:ffff/64 Scope:Global
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:59 errors:0 dropped:0 overruns:0 frame:0
          TX packets:10 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:9877 (9.8 KB)  TX bytes:1312 (1.3 KB)

(output
                truncated)

```


- c. Enter the **ping** command—specifying a known URL—to verify that you have internet connectivity.

```
root@VLAB1:~# ping example.com
```

3. Install the vSLX debian package.

```
root@VLAB1:/slxos-dist# sudo dpkg -i /slxos-dist/vslx_2.0.0.deb
Selecting previously unselected package vslx.
(Reading database ... 13616 files and directories currently installed.)
Preparing to unpack ./vslx_2.0.0.deb ...
Unpacking vslx (2.0.0) ...
Setting up vslx (2.0.0) ...
```

4. Run the post-installation script.

```
root@VLAB1:~# /VM/postinst-setup.sh
Get:1 http://security.ubuntu.com/ubuntu xenial-security InRelease [102 kB]
Hit:2 http://archive.ubuntu.com/ubuntu xenial InRelease
Get:3 http://archive.ubuntu.com/ubuntu xenial-updates InRelease [102 kB]
Fetched 204 kB in 1s (202 kB/s)
Reading package lists... Done
Reading package lists... Done
```

After successful installation, proceed to [Up-and-running example](#) on page 29.

Reinstalling vSLX in a container

After reinstalling the xlc bash scripts, perform this task to upgrade or downgrade a vSLX container installation—preserving your vSLX objects and configurations.



Note

This flow installs new vSLX and SLX-OS versions—while maintaining support for existing and new objects of currently installed SLX-OS versions. But if you need a clean upgrade, refer to [Uninstalling vSLX completely](#) on page 26.

1. In vShell, make sure that all devices (chassis and workstations) are powered off.

```
(vsh) poweroff ch_9540_01
```

2. Make sure that any container VMs are closed.
3. Verify that all is ready for vSLX reinstallation, as follows:
 - a. List the new SLX distribution directory.

```
root@VLAB1:~# ls /slxos-dist
extreme-lxc_2.1.0.deb  slxos18r.2.00  slxos18r.2.00.tar.gz  vslx_2.1.0.deb
vslx2.1.0.tar.gz
```



Note

You can also list included directories.

- b. Enter the **ping** command—specifying a known URL—to verify that you have internet connectivity.

```
root@VLAB1:~# ping example.com
```

4. Uninstall the current vSLX distribution—preserving your vSLX objects and configurations.

```
root@VLAB1:~$ sudo dpkg -r vslx
```

5. Install the vSLX debian package.

```
root@VLAB1:/slxos-dist# sudo dpkg -i /slxos-dist/vslx_2.1.0.deb
Selecting previously unselected package vslx.
```

```
(Reading database ... 13616 files and directories currently installed.)
Preparing to unpack ./vslx_2.1.0.deb ...
Unpacking vslx (2.1.0) ...
Setting up vslx (2.1.0) ...
```

6. Run the post-installation script.

```
root@VLAB1:~# /VM/postinst-setup.sh
Get:1 http://security.ubuntu.com/ubuntu xenial-security InRelease [102 kB]
Hit:2 http://archive.ubuntu.com/ubuntu xenial InRelease
Get:3 http://archive.ubuntu.com/ubuntu xenial-updates InRelease [102 kB]
Fetched 204 kB in 1s (202 kB/s)
Reading package lists... Done
Reading package lists... Done
```

7. On the host, stop and restart the container.

```
vlab1_user@ubuntu:~$ sudo xlxc stop VLAB1
vlab1_user@ubuntu:~$ sudo xlxc start VLAB1
```

8. Verify that vSLX and Extreme-lxc are installed.

```
user@ubuntu1:~$ dpkg --get-selections | grep vslx
vslx                                install
user@ubuntu1:~$ dpkg --get-selections | grep extreme-lxc
extreme-lxc                         install
```

Uninstalling vSLX completely

Use this task to completely uninstall a vSLX instance, including your vSLX objects and configurations.



Note

The prompts below follow [Installing a vSLX lab on a host](#) on page 21. Following [Container installation \(multiple vSLX labs\)](#) on page 22, log in to the container.

1. Although this task automatically closes any vSLX VMs, close any other VMs.
2. Log in to vShell.

```
user@ubuntu:~$ vsh
```

3. Power off all vSLX devices.

```
(vsh) poweroff ch_9540_01
```

4. Delete all vSLX devices.

```
(vsh) delete chassis ch_9540_01
```

5. Delete all vSLX templates.

```
(vsh) delete template 9540_templ
```

6. If needed, undo the changes to `/etc/network/interfaces`.
7. Exit vShell.

```
(vsh) exit
```

8. Uninstall the vSLX distribution.

```
user@ubuntu:~$ sudo /VM/uninstall.sh
```

9. If needed, delete users that you created.

```
user@ubuntu:~$ sudo deluser testuser
```



vShell Basics

[Overview of vShell entities on page 27](#)

[Scaling Considerations on page 28](#)

[Up-and-running example on page 29](#)

[Basic vShell Tasks on page 31](#)

Overview of vShell entities

Extreme Virtual Shell (vShell) commands, objects, and relationships enable you to create and manage supported virtual devices and networks.

The vShell environment

Virtual Shell (vShell) is a Python-based interactive environment that enables you to create and manage virtual system instances.



Note

For documentation of vShell commands, refer to [Appendix A: vShell Command Reference](#) on page 48.

Templates and chassis

A *template* object is a virtual image of an Extreme SLX-OS device. A *chassis* is a bootable instance of a template that represents a device.

The current vSLX version supports SLX 9540 template types:

Chassis-creation flow is as follows:

1. Create named templates from one or both of the supported template types.
2. Create one or more chassis (devices), each specifying a named template.

Workstations

You can also create *workstations* (virtual PCs). In general, you virtually connect workstations to virtual devices (chassis) that you created.

The current version supports only workstations running Ubuntu Linux.

Bridges

You can connect a port to no more than one other port. A bridge enables you to connect multiple ports, forwarding traffic received by one port to all ports connected to the bridge.



Note

A virtual bridge does not learn MAC addresses.

Probes

To connect a probe to an entity—a device port, a bridge, or a host interface—use the **link -probe** command. Once the connection is created, traffic to and from the entity is forwarded to the probe. You can use x86 network tools to monitor probed traffic.

Links

A link is a virtual cable connecting two ports, a port and an entity, or two entities.

Scaling Considerations

This section contains information that helps you estimate CPU, RAM, and disk requirements for vSLX entities.

This information is valid for both installation options:

- [Installing a vSLX lab on a host](#) on page 21
- [Container installation \(multiple vSLX labs\)](#) on page 22



Note

For server CPU, memory, and storage specifications, refer to [vSLX server requirements](#) on page 15.

Resource requirements

The following table lists virtual CPU (vCPU), memory, and storage requirements needed to emulate SLX-OS entities.

Table 9: Resources required for vSLX entities

Entity	CPU cores	Virtual memory (RAM)	Min. disk size	Max. disk size
SLX 9540 template	—	—	9 GB	9 GB
SLX 9540 instance	1	8 GB	1 GB	25 GB

Memory swapping

If your storage technology is solid-state drive (SSD)—rather than rotational hard-disk drive (HDD)—memory swapping into the SSD can reduce the physical "Virtual memory (RAM)" values in the preceding table. For example, a swap space of 32 GB supports four additional SLX 9540 instances.

Virtual machines (VMs)

One Virtual Machine (VM) can emulate one virtual device. vSLX uses several methodologies to optimize resource utilization among VMs:

- Space efficiency: QEMU snapshot technology is used for chassis (virtual device) image creation from a template. When a chassis is created, the initial storage size is less than 200 KB. As a chassis is configured, the disk size increases.
- Memory efficiency: QEMU and kernel same-page merging (KSM) support memory de-duplication among VMs, which enables sharing memory for duplicate blocks. These technologies reduce memory requirements by up to 60 percent.
- CPU over-provisioning: When vSLX is used for normal control-plane traffic, CPU consumption does not exceed two vCPUs for each VM. This efficiency allows for flexibility in over-subscribing CPU resources, allowing vSLX to scale well.

SLX 9540 scaling calculations

The following equations estimate optimized server resources needed for virtual SLX 9540 devices (each SLX 9540 instance is one VM):

- $CPU_{SLX\ 9540}$ (number of cores) = $1 * \langle \text{number-of-VMs} \rangle$
- $MEM_{SLX\ 9540}$ (GB) = $8 * \langle \text{number-of-VMs} \rangle$
- $STORAGE_{SLX\ 9540}$ (GB) = $TemplateSize_{SLX\ 9540} + (2 * \langle \text{number-of-VMs} \rangle)$

The following examples estimate the server resources needed to emulate 10 SLX 9540 instances:

- CPU = $1 * 10 = 10$ cores
- MEM = $8 * 10 = 80$ GB
- STORAGE = $6 + (2 * 10) = 26$ GB

Up-and-running example

This example logs you in to vShell, creates a template and two linked devices; turns on, connects to, logs in to the devices, and defines IP addresses; and logs out and disconnects from the devices.

1. From Linux, log in to vShell.

- On a host-based vSLX installation:

```
user@ubuntu:~$ vsh
For help anytime, type '?'
When you're done, type '^D'
(vsh)
```

- On a container-based vSLX installation:

```
root@VLAB1:~# vsh
Bridge address : 10.0.0.64/24
Default gateway : 10.0.0.1
For help anytime, type '?'
```

```
When you're done, type '^D'
(vsh)
```

- Enter the **create template** command.

```
(vsh) create template TMPL1-AV slx9540 /slxos-dist
Building SLX9540 Image ...

***** Created VM PB4000 *****
PB4000 created.
```

- To verify template creation, enter the **show templates** command.

```
(vsh) show templates
Name           Type           SLXOS Version   Snapshots
.....
TMPL1-AV      slx9540      slx-os17r.2.00  0
```

- Create two virtual SLX 9540 devices, using the **create chassis** command.

```
(vsh) create chassis av1 1U TMPL1-AV
Domain PB_1_av1 defined from /VM/templates/snapsh/av1/BVM_PB_SIM_1.xml
(vsh) create chassis av2 1U TMPL1-AV
Domain PB_1_av2 defined from /VM/templates/snapsh/av2/BVM_PB_SIM_1.xml
```

- Create a Linux workstation, using the **create workstation** command.

```
(vsh) create workstation h1 linux
Converting qcow2 image to raw format.
Please wait a few seconds...Done
/VM/templates/host/snapsh/h1/HOST_VM_DEF_S.xml
Domain h1 defined from /VM/templates/host/snapsh/h1/HOST_VM_DEF_S.xml
```

- Enter the **show system** command.

```
(vsh) show system
Device Name           Type           Power
.....
av1                   SLX 9540      off
av2                   SLX 9540      off
h1                    VPC           off
```

- Enter the **poweron** command to turn on the devices.

```
(vsh) poweron av1
The device is powered on successfully (1)
(vsh) 2 poweron av2
The device is powered on successfully (1)
(vsh) poweron h1
```

- Enter the **show system** command.

```
Device Name           Type           Power
.....
av1                   SLX 9540      on
av2                   SLX 9540      on
h1                    VPC           on
```

- Create a link between the devices.

```
(vsh) create link av1 0/1 av2 0/1
```



Note

If you were to run **show link** and **show link status** at this point, they would display "Down" and "Both sides down", respectively. The link will be up only after you log in to SLX-OS and configure av1 0/1 and av2 0/1, as described below.

10. Connect to one of the devices, log in to SLX-OS, and assign an IP address.

```
(vsh) connect av1
SLX-OS (SLX)SLX login: admin
password: password
device# configure terminal
device(config)# interface ethernet 0/1
device(config-if-eth-0/1)# ip address 10.0.0.1/24
device(config-if-eth-0/1)# no shut
```

11. Log out from the first device and return to vShell.

```
device(config-if-eth-0/1)# end
device# exit
SLX-OS (SLX) Ctrl-J
(vsh)
```

12. Connect to the second device, log in to SLX-OS, and assign an IP address.

```
(vsh) connect av2
SLX-OS (SLX)SLX login: admin
password: password
device# configure terminal
device(config)# interface ethernet 0/1
device(config-if-eth-0/1)# ip address 10.0.0.2/24
device(config-if-eth-0/1)# no shut
```

13. Log out from the second device and return to vShell.

```
device(config-if-eth-0/1)# end
device# exit
SLX-OS (SLX) Ctrl-J
(vsh)
```

14. To verify the link between the two devices, enter the **show link** command.

```
(vsh) show link
Name          Port          Name          Port          State
.....
av1           0/1    <-->  av2           0/1          Up
```

15. To turn off the devices, enter the **poweroff** command.

```
(vsh) poweroff av1
Stopping gos servers for chassis av1
The device is powered off successfully (1)
(vsh) poweroff av2
Stopping gos servers for chassis av2
The device is powered off successfully (1)
(vsh) poweroff h1
```

Basic vShell Tasks

Perform these tasks to log in to vShell, access CLI Help, and configure virtual devices.

Starting vShell



Note

If needed, you can open multiple vShell sessions from different Linux shells.

To start vShell, enter **vsh**.

```
$ vsh
For help anytime, type '?'
```

```
When you're done, type '^D'  
(vsh)
```

Displaying vShell Keyboard Shortcuts

To display vShell keyboard shortcuts, enter **help**.

```
(vsh) help  
? - for help  
Tab - auto fill  
^P - previous command  
^N - next command  
^D - end of the session  
^A - move cursor to the beginning  
^E - move cursor to the end  
^W - delete a word  
^U - delete from the beginning  
^K - delete to the end
```

Displaying vShell commands

To display available commands, enter a question mark (?).

```
(vsh) ?  
clear          Clear  
connect        Connect console  
create         Provisioning  
delete         Destroy  
exit           Exit shell  
help           Get help  
list           Show running devices  
poweroff       Power off device  
poweron        Power on device  
show           Show info  
start          Power on and connect  
system         Bridge daemon control
```

Displaying Command Options

To display command parameters, type the command, press the Space key, and then type a question mark (?).

```
(vsh) create chassis link ?  
-bridge        Bridge  
-interface      Interface  
-probe          Probe  
String         Device name
```

Creating a Template

To create a device template, enter the **create template** command.



Note

For details, refer to [create template](#) on page 57.

The following example shows how to create a required template type.

```
(vsh) create template 9540_tmpl slx9540 /slxos-dist
```

Creating a Virtual Device

To create a virtual device, enter the **create chassis** command.



Note

For details, refer to [create chassis](#) on page 53.

Deleting a Virtual Device

To delete a virtual device, enter the **delete chassis** command. The following example includes two devices, test1 and test2. After delete test1, the **show chassis** command displays only test2.

```
(vsh) show chassis
Chassis Name          Type          Power
.....
test1                 SLX 9540     on
test2                 SLX 9540     off

(vsh) delete chassis test1

(vsh) show chassis
Chassis Name          Type          Power
.....
test2                 SLX 9540     off
```

Turning on and Turning off a Virtual Device

To turn on a virtual device, enter the **poweron** command. To turn off a virtual device, enter the **poweroff** command.

```
(vsh) show system
Device Name          Type          Power
.....
av1                  SLX 9540     off
av2                  SLX 9540     off
host1                VPC          off
host2                VPC          off

(vsh) poweron av2

(vsh) show system
Device Name          Type          Power
.....
av1                  SLX 9540     off
av2                  SLX 9540     on
host1                VPC          off
host2                VPC          off
```

Creating a Virtual Workstation

To create a virtual PC workstation, enter the **create workstation** command.

```
(vsh) create workstation linux_ws_01 linux
```

Connecting to a Virtual Device

To connect the current `vsh` session to the console of a virtual workstation or chassis, enter the **connect** command.

```
(vsh) connect test1
```

```
SLX-OS (SLX)
```

```
SLX login:
```

Disconnecting from a Virtual Device

To disconnect from a vSLX virtual device, press **Ctrl-J**. (Press the **J** key while holding down the **Ctrl** key.)



Virtual Networks

[Overview of virtual networks](#) on page 35

[Links](#) on page 35

[Creating and linking a bridge](#) on page 36

[Creating and linking a probe](#) on page 37

[Linking across vSLX instances with tunnels](#) on page 37

[Linking to a Real Interface](#) on page 38

Overview of virtual networks

Under vSLX, you can create and connect virtual SLX devices that simulate a network.

You can also create and connect virtual workstations, bridges, probes, and tunnels, setting up a virtual network lab. Virtual networks are supported for both installation options:

- [Installing a vSLX lab on a host](#) on page 21
- [Container installation \(multiple vSLX labs\)](#) on page 22

Links

A link is a virtual network-cable that connects two device ports or other network entities.

The topics in this section explain how to create and delete links between two device ports. You can also link the following entities:

- Bridges
- Probes
- Workstations
- Tunnels

For configuration details of all these entities, refer to [create link](#) on page 54.

Creating Links Between Ports

Use this task to create a virtual Ethernet cable between two supported devices.

1. Create the first link.

```
(vsh) create link ch_9540_01 0/1 ch_9540_02 1/1
```

2. Create the second link.

```
(vsh) create link ch_9540_01 0/3 ch_9540_02 5/1
```

- (Optional) To verify link status, run the **show link status** command.

```
(vsh) show link status
Name          Port          Name          Port          Status
.....
ch_9540_01    0/1    <--> ch_9540_02    1/1          Right side down
ch_9540_01    0/3    <--> ch_9540_02    5/1          All up
```

Deleting links between ports

Use this task to delete virtual links between ports.

- Delete the first link.

```
(vsh) delete link ch_9540_01 0/1 ch_9540_02 1/1
```

- Delete the second link.

```
(vsh) delete link ch_9540_01 0/3 ch_9540_02 5/1
```

Creating and linking a bridge

Use this task to create a bridge and link it to devices.

A bridge enables you to connect a port to more than one additional port or other entity. The bridge forwards all traffic received by one entity to all other entities connected to the bridge.

- Enter the **create bridge** command.

```
(vsh) create bridge BR1
```

- Enter the required **create link** commands.

```
(vsh) create link -bridge BR1 RT1 4/10
(vsh) create link -bridge BR1 RT2 5/12
(vsh) create link -bridge BR1 host1 0/1
```

The following example shows how to create a bridge, link two devices and a host through the bridge, display the links, and display the bridge summary.

```
(vsh) create bridge BR1
(vsh) create link -bridge BR1 RT1 4/10
(vsh) create link -bridge BR1 RT2 5/12
(vsh) create link -bridge BR1 host1 0/1
(vsh) show link
Name          Port          Name          Port          State
.....
RT1           4/10    <--> BR1          [bridge]      Up
RT2           5/12    <--> BR1          [bridge]      Up
host1         0/1     <--> BR1          [bridge]      Up

(vsh) show bridge
Name          No.connections
.....
BR1           3
```

Creating and linking a probe

Use this task to create a probe and link it to devices.

When you link a probe to a network entity, traffic sent from or received by the entity is forwarded to the probe. You can then use host utilities like `tcpdump` to monitor the traffic.

1. Enter the **create probe** command.

```
(vsh) create probe PB2
```

2. Enter the required **create link** command.

```
(vsh) create link -probe PB1 RT3 1/4
```

The following example shows how to create a probe, link it to a device port, display the links, and display the probe summary.

```
(vsh) create probe PB2
vsh) create link -probe PB1 RT3 1/4
(vsh) show link
Name          Port          Name          Port          State
.....
RT1           1/2          <--> RT3         1/4           Up
RT3           1/4          <--> PB2         [probe]       Up
(vsh) show probe
Name          Target
.....
PB2           RT3          1/4
```

Linking across vSLX instances with tunnels

A pair of virtual UDP tunnels connects two vSLX instances—both of which are running vSLX—to form a larger virtual network.

Creating virtual tunnels enables you to emulate larger virtual labs distributed over two or more vSLX instances. These vSLX instances can be on:

- Multiple servers
- Separate containers, according to [Container installation \(multiple vSLX labs\)](#) on page 22
- Separate VMware virtual machines (VMs), according to [Appendix C: vSLX on VMware ESXi](#) on page 77

1. On vSLX-instance A, enter the **create tunnel** *tunnel-name peer-ip udp-port* command.

```
(vsh) create tunnel tu_serv_a_b 10.1.1.2 1000
```

2. On vSLX-instance A, create a link between a virtual device to a "port" of the tunnel you just created.

```
(vsh) create link ch_9540_01 0/1 tu_serv_a_b 1/1
```

3. On vSLX-instance B, enter the **create tunnel** *tunnel-name peer-ip udp-port* command.

```
(vsh) create tunnel tu_serv_b_a 10.1.1.1 1000
```



Note

Make sure that you specify the same *udp-port* as for the vSLX-instance A tunnel.

- On vSLX-instance B, create a link between a virtual device to a "port" of the tunnel you just created.

```
(vsh) create link ch_9540_101 0/5 tu_serv_b_a 1/1
```



Note

Make sure that you specify the same tunnel *slot / port* as for the vSLX-instance A link.

Linking to a Real Interface

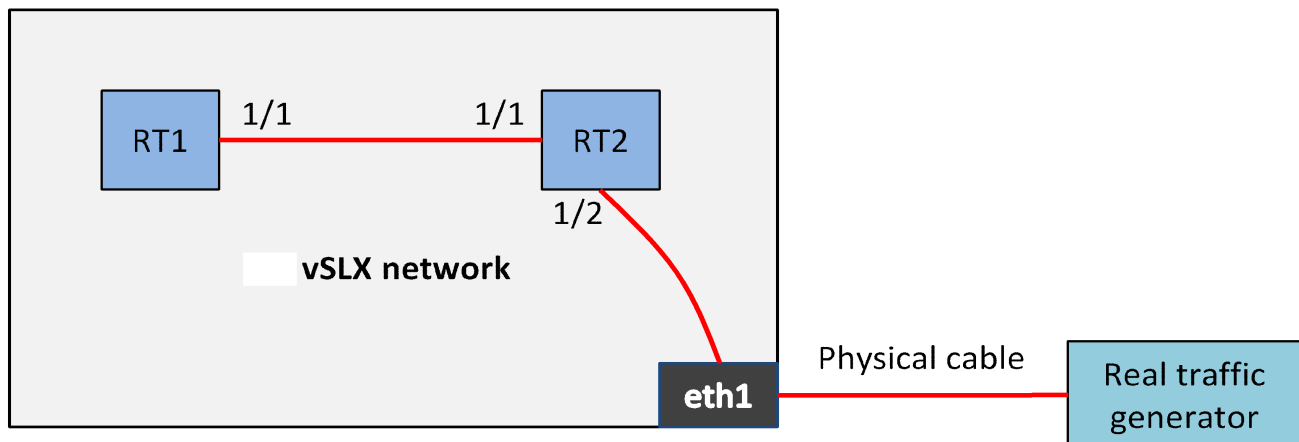
Perform this task to link from a virtual device on a vSLX network to a real port on the server running vSLX.



Note

A common use-case is to support connection of an external traffic generator into a virtual network.

The following diagram displays a virtual network with two routers, RT1 and RT2. You link RT1 and RT2 and link RT2 to a real port (eth1) on the server hosting the vSLX instance. You can then connect an external device to that port.



- Link RT1 and RT2.

```
(vsh) create link RT1 1/1 RT2 1/1
```

- Link RT2 to the real interface eth1.

```
(vsh) create link RT2 1/2 -interface eth1
```



IP Fabric Emulation

[IP Fabric reference topology](#) on page 39

[IP Fabric features tested](#) on page 39

IP Fabric reference topology

The following diagram illustrates a BGP-EVPN IP Fabric topology that you can emulate in vSLX.



Note

Although vSLX supports IP Fabric control-path functionality, data path is not supported.

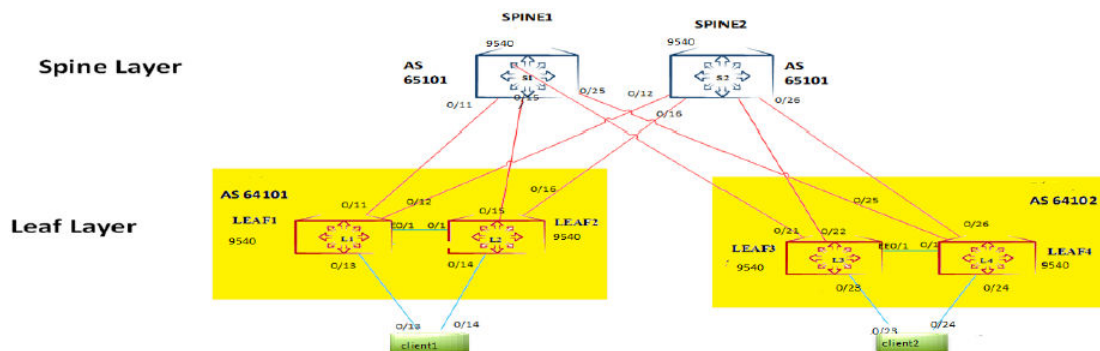


Figure 1: BGP-EVPN IP Fabric reference topology

Hardware requirements:

- CPU: 24 cores
- Memory: 132 GB
- Storage: 512 GB

IP Fabric features tested

The following IP Fabric control-plane and data-plane features have been tested for this release.

IP control-plane features tested

The following IP Fabric control-plane features have been tested for this release.

- IP Fabric control-plane
- L2VNI (VLAN and bridge-domain)

- L3VNI (VLAN and bridge-domain)

IP data-plane features tested

The following IP Fabric data-plane features have been tested for this release.

- L2VNI (VLAN and bridge-domain)
- L3VNI (VLAN and bridge-domain)



Layer 2 Exchange Emulation

[L2 Exchange reference topology](#) on page 41

[L2 Exchange features tested](#) on page 42

L2 Exchange reference topology

The following diagram illustrates a Layer 2 Exchange topology that you can emulate in vSLX.



Note

Although vSLX supports control-path functionality, data path is not supported.

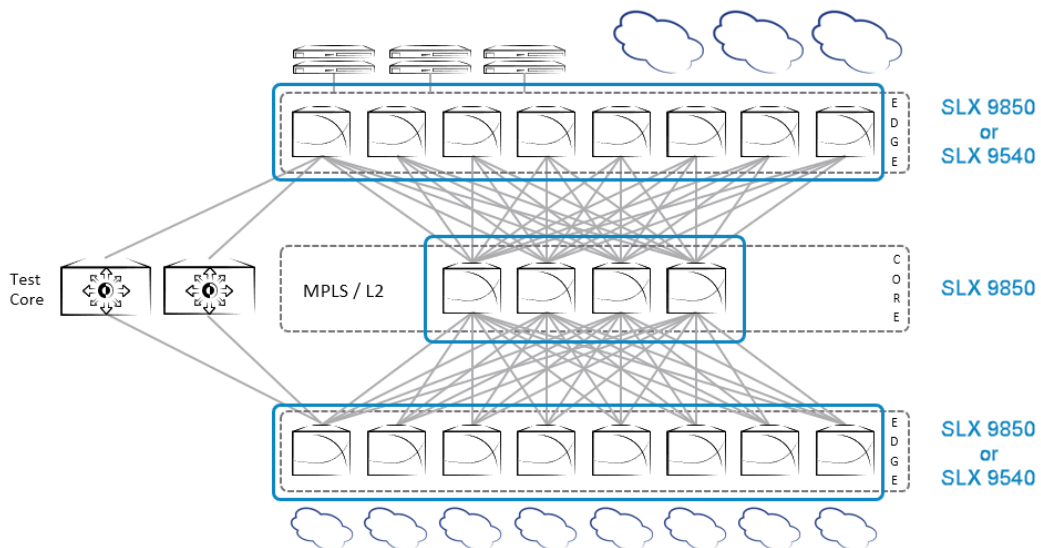


Figure 2: L2 Exchange reference topology

Key capabilities

The key capabilities supported are as follows:

- Layer 2—VLAN, bridge-domain (BD)
- Layer 3—Underlay with IPv4
- MPLS Control Plane—LDP, MPLS TE (RSVP-TE)
- VPLS and VLL Control Plane—LDP

L2 Exchange features tested

The following L2 Exchange features have been tested for this release.

VPLS features tested

The following Virtual Private LAN Service (VPLS) features have been tested for this release:

- VPLS PW bring-up with RSVP and OSPF as IGP
- Deletion and addition of MPLS config, bridge-domain with p2mp config, and tunnel config
- Verification of VPLS PW bring-up in raw ,tagged, and raw pass-through modes
- Verification of flapping IGP and RSVP sessions

VLL features tested

The following Virtual Leased Lines (VLL) features have been tested for this release:

- VLL PW bring-up with LDP and ISIS as IGP
- Deletion and addition of MPLS config, bridge-domain with p2p config, and tunnel config in VLL
- Verification of VLL PW bring-up in raw, tagged, and raw pass-through modes
- Verification of flapping IGP and LDP sessions

Known L2 Exchange issues

Ping is not currently supported under L2 Exchange.



Advanced vShell Features

[vShell commands from the Linux shell on page 43](#)

[User-developed commands and scripts on page 43](#)

vShell commands from the Linux shell

As an alternative to entering vShell commands from the vShell prompt, you can enter commands from the Linux shell.

The following Linux syntax is equivalent to `(vsh) show templates`:

```
root@your_user-Iatitude-E5540:/home/your_user/lxc/vslxCont# vsh show templates
```

User-developed commands and scripts

Because vShell was developed using the Python programming language, you can extend vShell with Python, creating commands and scripts.

Python requirements for extending vShell

The guidelines for using Python to modify your vShell implementation are as follows:

- Although previous experience in Python programming is helpful, experience in other high-level languages is enough to get you started.
- To help decide which editor or integrated development environment (IDE) to use, refer to <http://www.python.org>.
- The supported version of Python is 2.7.

Implementation flow for vShell extensions

This topic presents a top-level view of how to add a command to your vShell instance.

1. Open a Python editor or integrated development environment (IDE) and create a new `*.py` file.
2. Type the required `from vsh import` statement.

```
# Import Python functions from the vsh module
from vsh import output, cmdparser
```

3. Create the target function.

```
# Define the function that the command will call.
def hello(username):

    output("Hello %s, how are you doing?", username)
```

4. Create the command that calls the target function.

```
# cmdparser(<target-function>, <syntax>, <msg>)
cmdparser(hello, "hello:%s", "Say hello to someone")
```

5. Save the file and copy it to the ~/.vsh directory.

```
bash$ cp demo.py ~/.vsh
```

6. Start vShell.

```
$ vsh
For help anytime, type '?'
When you're done, type '^D'
(vsh)
```

7. Verify that the command you created is available.

```
(vsh) ?
clear          Clear
connect       Connect console
create        Provisioning
hello         Say hello to someone
delete        Destroy
exit          Exit shell
help          Get help
list          Show running devices
poweroff      Power off device
poweron       Power on device
show          Show info
start         Power on and connect
system        Bridge daemon control
```

8. Enter the command—with required parameters—and verify that it performs as expected.

```
(vsh) hello Mark
Hello Mark, how are you doing?
```



Note

The command you created will load every time you log in to this instance of vShell.

Syntax of the cmdparser() function

The `cmdparser(<target-function>, <syntax>, <msg>)` function creates commands that you can use in your vShell instance.

As indicated in [Implementation flow for vShell extensions](#) on page 43, `cmdparser()` requires a target function.

<target-function>

Specifies the name of the function called by `cmdparser()`.

```
<syntax> = "<vSLX-command-name>:<keyword>|<argument>:<keyword>|
<argument>: ..."
```

Enclose the `<syntax>` element in quotes ("). The first word of the `<syntax>` element is the name of the vShell command you are creating, followed by the required keywords and arguments.

There is no special format for keywords. For each argument, you specify a type symbol, as follows:

Table 10: vShell data types

Data type	Type symbol
String	%s
Integer	%d
IP address	%i
MAC address	%m or %M
Interface port	%l

For example, the syntax of the vShell `create link` command is `"create:link:%s:%I%s:%I"`. The elements of this command are as follows:

- Command name—`create`
- Keyword—`link`
- Arguments (separate by colons)—`%s:%I%s:%I`

Sample CLI input by vShell user:

```
create link rt1 0/1 rt2 0/1
```

`<msg>`

Specifies the CLI Help displayed when a user types `?`.

Enclose the `<msg>` element in quotes (`"`).

```
"Creates a link between two entities."
```

Implementing a configuration script

This Python script creates the links in a network that includes a router, three workstations, a bridge, and a tunnel.



Note

Because vSLX configuration is persistent, this script is not needed for a single vSLX virtual lab. But it might be useful for setting up multiple networks to run multiple tests.

1. Open a Python editor or integrated development environment (IDE) and create a new `*.py` file.
2. Type the required `from vsh import` statement.

```
from vsh import m, cmdparser
# The function "m" defines a command executable in vShell.
```

3. Create the target function.

```
def demo():
```

```
m("create link RT3 1/1 host6 1")
m("create link RT3 1/2 -b BR1")
m("create link RT3 1/5 T2 1")
m("create link RT3 1/6 T2 2")
m("create link -b BR1 host7 1")
m("create link -b BR1 host30 1")
```

4. Create the `cmdparser (<target-function>, <syntax>, <msg>)` command that calls the target function.

```
cmdparser(demo, "demo_01", "Configures network of router, three workstations, bridge,
and tunnel.")
# The demo target function does not require any parameters.
```

5. Save the file and copy it to the `~/ .vsh` directory.

```
bash$ cp net_demo_01.py ~/.vsh
```

6. Start vShell.

```
$ vsh
For help anytime, type '?'
When you're done, type '^D'
(vsh)
```

7. Verify that the command you created is available.

```
(vsh) ?
clear                Clear
connect              Connect console
create               Provisioning
demo_01              Configures network of router, three workstations, bridge, and
tunnel.
delete               Destroy
exit                 Exit shell
help                 Get help
list                 Show running devices
poweroff             Power off device
poweron              Power on device
show                 Show info
start                Power on and connect
system               Bridge daemon control
```

8. Enter the command that you created.

```
(vsh) demo_01
```

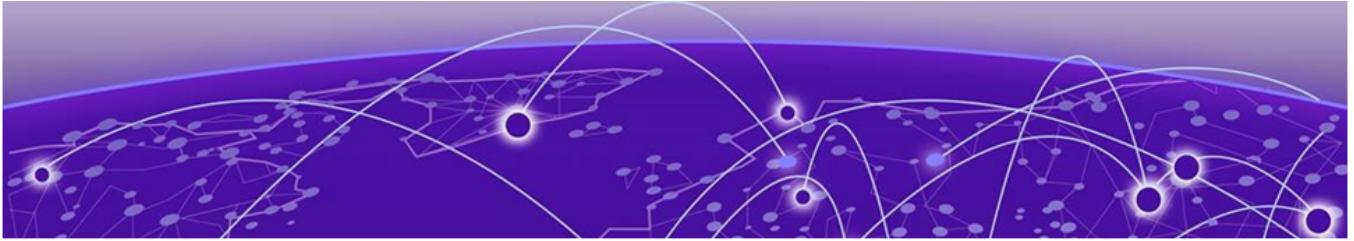


Note

The command you created will load every time you log in to this instance of vShell.

9. Enter the **show link** command to verify that your configuration was successful.

```
(vsh) show link
Name          Port      Name          Port      State
.....
RT3           1/1      <--> host6      0/1       Up
RT3           1/2      <--> BR1         [bridge]  Up
RT3           1/5      <--> T2           0/1       Up
RT3           1/6      <--> T2           0/2       Up
host30        0/1      <--> BR1         [bridge]  Up
host7         0/1      <--> BR1         [bridge]  Up
```



Guidelines for isolating issues to vSLX

To isolate issues to vSLX (as opposed to SLX-OS), check if basic IP connectivity (ping) works over the links:

- Boot up using simple L2 and L3 configuration and make sure that interfaces come up and that **ping** works.
- Use vsh commands to make sure that relevant nodes are powered on and that the console connection works.
- Links require both SLX-OS interface-level configuration and vsh link-creation. If links are down check both SLX-OS and vsh to make sure that the configuration is correct.

General guideline

If basic connectivity does not work, it is most likely a vSLX issue. Otherwise, it is most likely an SLX-OS issue.



Appendix A: vShell Command Reference

[clear debug](#) on page 49
[connect](#) on page 50
[create bridge](#) on page 51
[create chassis](#) on page 53
[create link](#) on page 54
[create probe](#) on page 56
[create template](#) on page 57
[create tunnel](#) on page 58
[create workstation](#) on page 59
[delete](#) on page 60
[help](#) on page 62
[list](#) on page 63
[poweroff](#) on page 64
[poweron](#) on page 65
[show](#) on page 66
[show debug](#) on page 68
[start](#) on page 69
[system](#) on page 70
[vsh](#) on page 71

clear debug

Clears all troubleshooting information and resets all counters to zero.

Syntax

```
clear debug
```

Command Default

Debug information is not deleted.

Modes

vShell prompt (vsh)

Usage Guidelines

There is no software limit to the debug cache.

This command does not have a **no** form.

Examples

The following example shows how to run the **clear debug** command.

```
(vsh) clear debug  
(vsh)
```

connect

Connects the current `vsh` session to the console of a virtual workstation or chassis.

Syntax

```
connect device-name
```

Command Default

The console is not connected to any device.

Parameters

device-name

Specifies a virtual device.

Modes

vShell prompt (`vsh`)

Usage Guidelines

Before you connect to an entity, you must turn it on, using the **poweron** command.

To turn on a device and connect to it with a single command, refer to [start](#) on page 69.

To disconnect the current `vsh` session from the device console, press **Ctrl-J**.

To log in to vShell and run this command, enter the `user@ubuntu:~$ vsh connect` command.

Examples

The following example shows how to connect the current `vsh` session to a device console.

```
(vsh) connect dev_01  
  
SLX-OS (SLX)  
SLX login:
```

The following example shows the error you get if you try to connect to a device that is turned off.

```
(vsh) connect h1  
h1: the device is powered off
```

create bridge

Creates a virtual bridge to connect multiple ports and devices.

Syntax

```
create bridge bridge-name
```

Command Default

No bridges are defined.

Parameters

bridge-name

Specifies a unique name for the bridge. A valid name must begin with an alphanumeric character. No special characters are allowed, except for the underscore (_) and hyphen (-).

Modes

vShell prompt (vsh)

Usage Guidelines

A bridge forwards all traffic received by one port to all other entities connected to the bridge.

To connect as many entities as required to the bridge, use the **link bridge** command.

To log into vShell and run this command, enter the `user@ubuntu:~$ vsh create bridge` command.

To delete a bridge, use the **delete bridge** command.

Examples

The following example shows how to create a bridge.

```
(vsh) create bridge bridge_01
```

The following example shows how to create a bridge, link two routers and a host through the bridge, display the links, and display the bridge summary.

```
(vsh) create bridge BR1
(vsh) create link -bridge BR1 RT1 4/10
(vsh) create link -bridge BR1 RT2 5/12
(vsh) create link -bridge BR1 host1 0/1
(vsh) show link
Name          Port          Name          Port          State
.....
RT1           4/10  <--> BR1       [bridge]      Up
RT2           5/12  <--> BR1       [bridge]      Up
host1        0/1    <--> BR1       [bridge]      Up
```

```
(vsh) show bridge
Name           No.connections
.....
BR1            3
```

create chassis

Creates a bootable virtual device from a vSLX template.

Syntax

```
create chassis chassis-name 1U template-name
```

Command Default

No chassis are defined.

Parameters

chassis-name

Specifies a unique name for the device. A valid name must begin with an alphanumeric character. No special characters are allowed, except for the underscore (_) and hyphen (-).



Note

Because chassis name-space is common to workstation name-space and tunnel name-space, we recommend naming conventions that prevent conflict.

1U

Specifies a virtual device. Make sure that you enter "1U" and not "1u".

template-name

Specifies the template.

Modes

vShell prompt (vsh)

Usage Guidelines

A chassis represents a virtual device.

To log into vShell and run this command, enter the `user@ubuntu:~$ vsh create chassis` command.

To delete a device, use the **delete chassis** command.

Examples

The following example shows how to create a virtual device from the template.

```
(vsh) create chassis ch_9540_01 1U 9540_temp1
```

create link

Emulates a cable connection for traffic between two entities.

Syntax

```
create link first-entity second-entity  
create link chassis-name slot / port second-entity  
create link workstation-name slot / port second-entity  
create link tunnel-name slot / port second-entity  
create link -bridge bridge-name second-entity  
create link -interface interface-name second-entity  
create link -probe probe-name second-entity
```

Command Default

No links exist.

Parameters

first-entity

Specifies the first device port, host-name, bridge, probe, or tunnel.

second-entity

Specifies the second device port, host-name, bridge, probe, or tunnel.

chassis-name

Specifies the SLX device for linking.

slot

Specifies a valid slot number. Must be 0 if the device does not contain slots.

port

Specifies a valid port number.

workstation-name

Specifies a workstation for linking.

slot

Specifies a valid slot number. Must be 0 if the device does not contain slots.

port

Specifies a valid port number.

tunnel-name

Specifies a tunnel interface for linking.

slot

Specifies a valid slot number. Values range from 0 through 15.

port

Specifies a valid port number. Values range from 1 through 4095.

-bridge *bridge-name*

Specifies a bridge for linking. You can abbreviate **-bridge** with **-b**.

-interface *interface-name*

Specifies a physical interface on the x86 server on which vSLX is running. You can abbreviate **-interface** with **-i**.

-probe *probe-name*

Specifies a probe for linking. You can abbreviate **-probe** with **-p**.

Modes

vShell prompt (vsh)

Usage Guidelines

To log into vShell and run this command, enter the `user@ubuntu:~$ vsh create link` command.

You can use the **create link -interface** *interface-name* option to connect a port on the x86 server to the virtual network you created. You can use this option to feed traffic-generator input into the virtual network.

In general, you link an entity from one server to an entity on another server through tunnel devices that are connected between the servers. (Tunnels are implemented in pairs between the two servers.)

To delete a link, use the **delete link** command.

Examples

The following example shows how to connect two ports on two virtual devices.

```
(vsh) create link ch_9540_01 0/1 ch_9540_02 1/1
(vsh) create link ch_9540_01 0/3 ch_9540_02 5/1
```

The following example creates a link connecting port 1/1 on router RT1 to port 1/1 on router RT2 and a link connecting port 1/2 on router RT2 to an x86 server port with hostname "eth1".

```
(vsh) create link RT1 1/1 RT2 1/1
(vsh) create link RT2 1/2 -interface eth1 1
```

create probe

Creates a virtual probe.

Syntax

```
create probe probe-name
```

Command Default

No probes are defined.

Parameters

probe-name

Specifies a unique name for the probe. A valid name must begin with an alphanumeric character. No special characters are allowed, except for the underscore (_) and hyphen (-).

Modes

vShell prompt (`vsh`)

Usage Guidelines

To connect a probe to an entity—a device port, a bridge, or a host interface—use the **link probe** command. Once the connection is created, traffic to and from the entity is forwarded to the probe.

To log into vShell and run this command, enter the `user@ubuntu:~$ vsh create probe` command.

To delete a probe, use the **delete probe** command.

Examples

The following example shows how to create a probe, link it to a device port, display the links, and display the probe summary.

```
(vsh) create probe PB2
(vsh) create link -probe PB1 RT3 1/4
(vsh) show link
Name          Port          Name          Port          State
.....
RT1           1/2          <--> RT3         1/4           Up
RT3           1/4          <--> PB2         [probe]       Up
(vsh) show probe
Name          Target
.....
PB2           RT3          1/4
```


create template

Creates a template for virtual instances of the supported SLX devices.

Syntax

```
create template template-name { slx9540 } vslx-distribution-path
```

Command Default

No templates are defined.

Parameters

template-name

Specifies a unique name for the template. A valid name must begin with an alphanumeric character. No special characters are allowed, except for the underscore (_) and hyphen (-).

slx9540

Specifies a template for virtual SLX 9540 instances.

vslx-distribution-path

Specifies the full path to the vSLX distribution installed on the device. The first character must be a slash (/). If the image is on a remote server, only NFS mount is supported.

Modes

vShell prompt (`vsh`)

Usage Guidelines

To log into vShell and run this command, enter the `user@ubuntu:~$ vsh create template` command.

To delete a template, use the **delete template** command.

Examples

The following example creates a SLX 9540 template.

```
(vsh) create template 9540_temp1 slx9540 /slxos-dist
```

create tunnel

A virtual tunnel connects two x86 servers—both of which are running vSLX—to form a larger virtual network.

Syntax

```
create tunnel tunnel-name peer-ip udp-port
```

Command Default

No tunnels exist.

Parameters

tunnel-name

Specifies a unique name for the tunnel. A valid name must begin with an alphanumeric character. No special characters are allowed, except for the underscore (_) and hyphen (-).



Note

Because tunnel name-space is common to workstation name-space and chassis name-space, we recommend naming conventions that prevent conflict.

peer-ip

Specifies the IP address of the other server.

udp-port

Specifies the port number, which must be identical on both servers and available on both servers.

Modes

vShell prompt (`vsh`)

Usage Guidelines

You need to create two tunnels, one on each server. A pair of tunnels can support thousands of virtual channels between the servers.

To delete a tunnel, use the **delete tunnel** command.

Examples

The following example uses a pair of virtual tunnels to connect virtual devices on two x86 servers—both of which are running vSLX—to form a larger virtual network.

```
(vsh) create tunnel tu_serv_a_b 10.1.1.2 1000
(vsh) create link ch_9540_01 0/1 tu_serv_a_b 1/1
(vsh) create tunnel tu_serv_b_a 10.1.1.1 1000
(vsh) create link ch_9540_101 0/5 tu_serv_b_a 1/1
```

create workstation

Creates a virtual PC workstation.

Syntax

```
create workstation workstation-name linux
```

Command Default

No workstations are defined.

Parameters

workstation-name

Specifies a unique name for the virtual workstation. A valid name must begin with an alphanumeric character. No special characters are allowed, except for the underscore (_) and hyphen (-).



Note

Because workstation name-space is common to chassis name-space and tunnel name-space, we recommend naming conventions that prevent conflict.

linux

Specifies a workstation running Ubuntu Linux.

Modes

vShell prompt (`vsh`)

Usage Guidelines

The current version only supports virtual Ubuntu-Linux workstations.

To log into vShell and run this command, enter the `user@ubuntu:~$ vsh create workstation` command.

To delete a workstation, use the **delete workstation** command.

Examples

The following example shows how to create a virtual Linux workstation.

```
(vsh) create workstation linux_ws_01 linux
```

delete

Deletes virtual entities created under vSLX.

Syntax

```

delete { bridge bridge-name | chassis chassis-name | probe probe-name |
  tunnel tunnel-name | workstation workstation-name }

delete template template-name [ force ]

delete link -all

delete link device entity-name

delete link entity-name slot / port { -bridge bridge-name | -interface
  interface-name | -probe probe-name | entity-name slot / port }

delete link -bridge bridge-name { -interface interface-name | -probe
  probe-name | entity-name slot / port }

delete link -interface interface-name { -bridge bridge-name | -probe
  probe-name | entity-name slot / port }

delete link -probe probe-name { -bridge bridge-name | -interface
  interface-name | entity-name slot / port }

```

Parameters

bridge *bridge-name*

Specifies a bridge for deletion.

chassis *chassis-name*

Specifies the SLX device for deletion.

probe *probe-name*

Specifies a probe for deletion.

tunnel *tunnel-name*

Specifies a tunnel for deletion. Deleting a tunnel also delete links connected to the tunnel.

workstation *workstation-name*

Specifies a workstation for deletion.

template *template-name*

Specifies a template for deletion.

force

Delete the template and all chassis (devices) created from it.

link

Specifies a link for deletion.

-all

Deletes all links in the current vShell instance.

device *entity-name*

Deletes all links between the specific chassis, workstation, or tunnel and all other entities.

entity-name slot / port

Deletes a link between the specific chassis port, workstation, or tunnel and the following specified entity. For a workstation, the *slot* value is 0.

-bridge *bridge-name*

Deletes links between the specific bridge and the additional specified entity.

-interface *interface-name*

Deletes links between the specific physical interface on the vSLX x86 server and the following specified entity.

-probe *probe-name*

Deletes links between the specific probe and the following specified entity.

Modes

vShell prompt (`vsh`)

Usage Guidelines

To log into vShell and run this command, enter the `user@ubuntu:~$ vsh delete` command.

If you delete a device, all of the links connected to the device are deleted.

Examples

The following example shows how to delete a link between two virtual devices.

```
(vsh) delete link ch_9540_01 0/3 ch_9540_02 5/1
```

The following example shows how to delete all links.

```
(vsh) delete link -all
```

The following example shows how to delete a bridge and all links connected to it.

```
(vsh) delete bridge br_01
```

The following example shows how to delete a probe and all links connected to it.

```
(vsh) delete probe pr_01
```

The following example shows how to delete a tunnel and all links connected to it.

```
(vsh) delete tunnel tu_01
```

help

Displays the vShell keyboard shortcuts.

Syntax

help

Modes

vShell prompt (vsh)

Usage Guidelines

To log into vShell and display the vShell keyboard shortcuts, enter the `user@ubuntu:~$ vsh help` command.

Examples

The following example displays the keyboard shortcuts.

```
(vsh) help
? - for help
Tab - auto fill
^P - previous command
^N - next command
^D - end of the session
^A - move cursor to the beginning
^E - move cursor to the end
^W - delete a word
^U - delete from the beginning
^K - delete to the end
```

The following example shows how to log in to vShell and immediately display the keyboard shortcuts.

```
user@ubuntu:~$ vsh help
? - for help
Tab - auto fill
^P - previous command
^N - next command
^D - end of the session
^A - move cursor to the beginning
^E - move cursor to the end
^W - delete a word
^U - delete from the beginning
^K - delete to the end
```

list

Displays a list of virtual devices, specifying how many virtual machines (VM) are running on each device.

Syntax

```
list [ -all ]
```

Parameters

-all

Specifies to display all defined devices, including those with no VMs running.

Modes

vShell prompt (vsh)

Usage Guidelines

To displays all defined devices—including those with no VMs running—include the **-all** keyword.

Examples

The following example displays all running devices.

```
(vsh) list
Name                No.running VMs
.....
a1                   1
abc                  1
abd                  1
abe                  1
```

The following example displays all defined devices.

```
(vsh) list -all
Name                No.running VMs
.....
a1                   1
a2                   0
a3                   0
abc                  1
abd                  1
abe                  1
```

poweroff

Emulates turning off a device.

Syntax

```
poweroff device-name
```

Command Default

The virtual status of all devices is turned off.

Parameters

device-name

Specifies the device name.

Modes

vShell prompt (`vsh`)

Usage Guidelines

To log into vShell and initially run this command, enter the `user@ubuntu:~$ vsh poweroff` command.

Examples

The following example shows how to turn off a virtual device.

```
(vsh) poweroff ch_9540_01
```


poweron

Emulates turning on a device.

Syntax

```
poweron device-name
```

Command Default

The virtual status of all devices is turned off.

Parameters

device-name

Specifies the device name.

Modes

vShell prompt (`vsh`)

Usage Guidelines

To log into vShell and run this command, enter the `user@ubuntu:~$ vsh poweron` command.

Examples

The following example show how to turn on a virtual device.

```
(vsh) poweron ch_9540_01
```

show

Displays various aspects of the virtual system configured under vSLX.

Syntax

```
show chassis [ chassis-name ]  
show link [ status ]  
show { bridge | probe | system | templates | tunnel | version }
```

Parameters

chassis

Displays details of all chassis or a specific chassis.

chassis-name

Displays details of a specific chassis.

link

Displays details of virtual cables between ports.

status

Also displays which sides of links are up or down.

bridge

Displays bridge connections.

probe

Displays probe names and targets.

system

Displays supported devices.

templates

Displays template names, types, corresponding SLX-OS versions, and the number of instances created.

tunnel

Displays tunnel names, peer IP addresses, ports, and statuses.

version

Displays the vShell version number.

Modes

vShell prompt (`vsh`)

Usage Guidelines

To log into vShell and run this command, enter one of the `user@ubuntu:~$ vsh show` commands.

Output

The **show system** command displays the following information:

Output field	Description
SLX 9540	Emulates an SLX 9540 switch.
VPC	Emulates a virtual PC running Linux.

Examples

To display details of all virtual devices, enter the **show chassis** command.

```
(vsh) show chassis
Chassis Name          Type          Power  Template
.....
a1                    SLX 9540     on     Avalanche
a2                    SLX 9540     on     Avalanche
```

To display details of all virtual devices, enter the **show system** command.

```
(vsh) show system
Device Name          Type          Power
.....
av1                  SLX 9540     off
av2                  SLX 9540     off
av3                  SLX 9540     off
av4                  SLX 9540     off
host1                VPC          off
host2                VPC          off
```

To display details of virtual cables between ports, enter the **show link** command.

```
(vsh) show link
Name          Port          Name          Port          State
.....
a1            0/1    <--> a2            0/1            Down
a1            0/5    <--> a2            0/5            Down
```

To display template names, types, corresponding SLX-OS versions, and the number of instances created, enter the **show templates** command.

```
(vsh) show templates
Name          Type          SLXOS Version  Snapshots
.....
aaa          slx9540     slx-os17r.2.01no  0
ava          slx9540     slx-os17r.2.01no  4
abc          slx9540     slx-os17r.2.01no  0
```

To display the vShell version, enter the **show version** command.

```
(vsh) show version
VShell Version 1.6.3-0
Copyright (c) Extreme Networks, Inc., 1996-2018
```

show debug

Displays vSLX troubleshooting information.

Syntax

```
show debug
```

Modes

vShell prompt (vsh)

Usage Guidelines

Diagnostic commands are developed and intended for specialized troubleshooting. Work closely with Extreme Networks technical support when running **debug** or **show system internal** commands and interpreting their results.

Examples

The following example displays a no-problem situation.

```
(vsh) show debug
Devices.....
Peers.....
a1[0.1] -- Int: Dev:
a2[0.1] -- Int: Dev:
a1[0.5] -- Int: Dev:
a2[0.5] -- Int: Dev:
Stats.....
Int a1[0.1] -- 0tx 0rx 0ttl
Int a2[0.1] -- 0tx 0rx 0ttl
Int a1[0.5] -- 0tx 0rx 0ttl
Int a2[0.5] -- 0tx 0rx 0ttl
Exception.....
0load 1linkscan 0devput 0devbind 0ifput
0sock 0rcvloop 0errpkt 0errkey 0server
```

start

Emulates connecting to a device and turning it on.

Syntax

```
start device-name
```

Command Default

The virtual status of all devices is turned off.

Parameters

device-name

Specifies the device name.

Modes

vShell prompt (vsh)

Usage Guidelines

For the specific device, the **start** command implements the **poweron** command and then instantaneously the **connect** command. The output of the **start** command displays the messages generated as the connected device boots up.

To log into vShell and run this command, enter the `user@ubuntu:~$ vsh start` command.

Examples

The following example displays typical output starting a device.

```
(vsh) start RT1
The device is powered on successfully (1)
Loading Linux 2.6.34.6 from partition 1 ...
[ 0.000000] this is quiet!
[ 0.000000] MTRR default type: write-back
[ 0.000000] MTRR fixed ranges enabled:
[ 0.000000] 00000-9FFFF write-back
[ 0.000000] A0000-BFFFF uncachable
[ 0.000000] C0000-FFFFFF write-protect
[ 0.000000] MTRR variable ranges enabled:
[ 0.000000] 0 base 00C0000000 mask FFC0000000 uncachable
[ 0.000000] 1 disabled
[ 0.000000] 2 disabled
[ 0.000000] 3 disabled
```

system

Starts, stops, or restarts the vShell bridge daemon, which forwards traffic between linked ports.

Syntax

```
system { start | stop | restart }
```

Parameters

start

Starts the vShell bridge daemon.

stop

Stops the vShell bridge daemon.

restart

Stops and then starts the vShell bridge daemon.

Modes

vShell prompt (vsh)

Usage Guidelines

The vShell bridge daemon starts automatically with vShell. The **system** command is reserved for troubleshooting and development.

Examples

The following example stops the vShell bridge daemon.

```
(vsh) system stop
```

vsh

Accesses the Extreme Virtual Shell (vShell), which enables you to create and manage virtual SLX-OS instances and networks.

Syntax

vsh

Modes

Ubuntu Linux on an x86 server

Usage Guidelines

If you append a vShell command to `user@ubuntu:~$ vsh`, you access vShell and the command immediately runs.

After you are logged in to vShell, to display the top-level vShell commands, press **?**.

To exit vShell, enter **exit** or press **Ctrl-D**.

Examples

(Host installation) The following example shows how to access vShell from Ubuntu Linux.

```
user@ubuntu:~$ vsh
For help anytime, type '?'
When you're done, type '^D'
(vsh)
```

The following example shows how to display the top-level vShell commands.

```
(vsh) ?
connect          Connect console
create           Provisioning
delete           Destroy
exit             Exit shell
help            Get help
list            Show running devices
poweroff         Power off device
poweron          Power on device
show            Show info
start           Power on and connect
```

The following example shows how to exit vShell back into Ubuntu Linux.

```
(vsh) exit
user@ubuntu:~$
```



Appendix B: Alternate Routines

[Multiple vSLX labs \(lxc\) on page 72](#)

Multiple vSLX labs (lxc)

(For the **lxc** flow only) Installing vSLX in Linux containers—rather than directly on the host—enables each user to have an independent virtual lab.

Linux Containers (LXC) support running multiple Linux systems on a control host using a single Linux kernel. We use *privileged* containers, created by root and running as root. After creating containers, you can install one vSLX virtual lab in each container.

Creating a container for vSLX (lxc)

(For the **lxc** flow only) This task is a prerequisite for installing each of the multiple instances of vSLX on the server.



Note

Although this legacy, unscripted flow is still supported, we recommend the scripted flow under [Container installation \(multiple vSLX labs\)](#) on page 22.

1. Log in to the x86 server as a user with sudo privileges.



Note

Although all sudo users can create and access all containers, our user in this flow is `vlab1_user`, created in [Creating additional Linux users](#) on page 17.

2. Install `lxc`, using the following commands, confirming prompts to continue.

```
vlab1_user@ubuntu:~$ sudo apt-get update
vlab1_user@ubuntu:~$ sudo apt-get install lxc
```


3. Enter the **lxc-create** command to create the container.

```
vlabl_user@ubuntu:~$ sudo lxc-create --name VLAB1 --template download --bdev dir -- --
dist ubuntu --release xenial --arch amd64 --force-cache --no-validate --server
images.linuxcontainers.org
```



Note

Because the previous example did not specify `-P <directory-path>`, the VLAB1 container is created in the default directory path, `/var/lib/lxc`. Command syntax is as follows:

```
sudo lxc-create [ -P <directory-path> ] --name <container-name> --template
download --bdev dir -- --dist ubuntu --release xenial --arch amd64 --force-
cache --no-validate --server images.linuxcontainers.org
```

4. Verify container creation.

```
vlabl_user@ubuntu:/slxos-dist$ sudo ls /var/lib/lxc
```

5. Create a `mkdev.sh` file and make it executable, as follows:

- a. Enter the **sudo vi** command.

```
vlabl_user@ubuntu:/slxos-dist$ sudo vi /var/lib/lxc/VLAB1/rootfs/root/mkdev.sh
```

- b. Copy the following content into `mkdev.sh`.

```
#!/bin/bash
#
# LXC autodev hook for Ubuntu 16 Container
#
# Some required device files for vSLX are
# not imported inside the container. So manually
# create them.
#
croot=${LXC_ROOTFS_MOUNT}
for i in `/usr/bin/seq 0 7`
do
    /bin/mknod $croot/dev/loop$i b 7 $i
done

mknod $croot/dev/loop-control c 10 237

for j in `/usr/bin/seq 0 11`
do
    /bin/mknod $croot/dev/dm-$j b 252 $j
done

/bin/mkdir -p $croot/dev/net
/bin/mknod $croot/dev/net/tun c 10 200

exit 0
```

- c. After you save and close the `mkdev.sh` file, make it executable.

```
vlabl_user@ubuntu:/slxos-dist$ sudo chmod +x /var/lib/lxc/VLAB1/rootfs/root/mkdev.sh
vlabl_user@ubuntu:/slxos-dist$ sudo ls -l /var/lib/lxc/VLAB1/rootfs/root/mkdev.sh
-rwxr-xr-x 1 root root 453 May  2 16:13 /var/lib/lxc/VLAB1/rootfs/root/mkdev.sh
```

6. Prepare the container `config` file, as follows:

- a. Enter the **sudo vi** command to open the file for editing.

```
vlabl_user@ubuntu:/slxos-dist$ sudo vi /var/lib/lxc/VLAB1/config
```

The initial `config` file created by running **lxc-create** displays:

```
# Template used to create this container: /usr/share/lxc/templates/lxc-download
# Parameters passed to the template: --dist ubuntu --release xenial --arch amd64
```

```
# --force-cache --no-validate --server images.linuxcontainers.org
# Template script checksum (SHA-1): 9748088977ba845f625e45659f305a5395c2dc7b
# For additional config options, please look at lxc.container.conf(5)

# Uncomment the following line to support nesting containers:
#lxc.include = /usr/share/lxc/config/nesting.conf
# (Be aware this has security implications)

# Distribution configuration
lxc.include = /usr/share/lxc/config/ubuntu.common.conf
lxc.arch = x86_64

# Container specific configuration
lxc.rootfs = /var/lib/lxc/VLAB1/rootfs
lxc.rootfs.backend = dir
lxc.utsname = VLAB1

# Network configuration
lxc.network.type = veth
lxc.network.link = lxcbr0 <-- Change it to br0
lxc.network.flags = up
lxc.network.hwaddr = 00:16:3e:b5:7d:b9
```

- b. Update the container config file with the values that you require. For example:

```
# Template used to create this container: /usr/share/lxc/templates/lxc-download
# Parameters passed to the template: --dist ubuntu --release xenial --arch amd64
# --force-cache --no-validate --server images.linuxcontainers.org
# Template script checksum (SHA-1): 9748088977ba845f625e45659f305a5395c2dc7b
# For additional config options, please look at lxc.container.conf(5)

# Uncomment the following line to support nesting containers:
#lxc.include = /usr/share/lxc/config/nesting.conf
# (Be aware this has security implications)

# Distribution configuration
lxc.include = /usr/share/lxc/config/ubuntu.common.conf
lxc.arch = x86_64

# Container specific configuration
lxc.rootfs = /var/lib/lxc/VLAB1/rootfs
lxc.rootfs.backend = dir
lxc.utsname = VLAB1

# Network configuration
lxc.network.type = veth
lxc.network.link = br0 <-- Changed from lxcbr0
lxc.network.flags = up
lxc.network.hwaddr = 00:16:3e:b5:7d:b9

# vSLX configuration <-- You need to add the following sections:
lxc.aa_profile = unconfined
lxc.cgroup.devices.allow = c 10:236 rwm
lxc.cgroup.devices.allow = b 252:* rwm
lxc.cgroup.devices.allow = b 7:* rwm

# Expose tun device
lxc.cgroup.devices.allow = c 10:200 rwm

# To export SLX-OS distribution directory into the container. Change paths as
# needed.
# The following implementation uses container /slxos directory as the container
# mount point to access the SLX-OS software distribution.
# The host-specific path for the same directory is /var/lib/lxc/VLAB1/rootfs/slxos.
# This implementation mounts your_user's home directory to the mount point /slxos.
```

```
lxc.mount.entry = /slxos-dist /var/lib/lxc/VLAB1/rootfs/slxos-dist none bind 0 0
# syntax: /<SLX-OS-Build-dir>/slxos-dist <directory-path>/<container-name>/rootfs/
slxos
# none bind 0 0

lxc.cgroup.devices.allow = c 10:237 rwm
lxc.autodev = 1
lxc.hook.autodev = ${LXC_ROOTFS_MOUNT}/root/mkdev.sh
```

- c. Save and close the file.
7. In the container, create an `slx-dist` directory.

```
vlab1_user@ubuntu:/slxos-dist$ sudo mkdir /var/lib/lxc/VLAB1/rootfs/slxos-dist
```

Installing vSLX in a container (lxc)

(For the **lxc** flow only) After preparing a Linux container, perform this task to install a vSLX virtual lab in the container.



Note

Although this legacy, unscripted flow is still supported, we recommend the scripted flow under [Container installation \(multiple vSLX labs\)](#) on page 22.

1. Start the container and attach to it.

```
vlab1_user@ubuntu:/slxos-dist$ sudo lxc-start -n VLAB1
vlab1_user@ubuntu:/slxos-dist$ sudo lxc-attach -n VLAB1
```

In these steps, the container name is "VLAB1".

2. Set a password for the default user "ubuntu".

```
root@VLAB1:/slxos-dist# passwd ubuntu
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
root@VLAB1:/slxos-dist# exit
exit
```

3. Log in to the container as "ubuntu".

```
user@ubuntu:/slxos-dist$ sudo lxc-console -n VLAB1
Connected to tty 1
Ubuntu 16.04.4 LTS VLAB1 pts/0

VLAB1 login: ubuntu
Password:
Welcome to Ubuntu 16.04.4 LTS (GNU/Linux 4.4.0-87-generic x86_64)

ubuntu@VLAB1:~$ sudo -i
[sudo] password for ubuntu:
root@VLAB1:~#
```



Note

To exit the container, type **Ctrl+a q**.

4. Become a root user.

```
ubuntu@VLAB1:~$ sudo -i
[sudo] password for ubuntu:
root@VLAB1:~#
```

5. Verify that all is ready for vSLX installation, as follows:

a. List the `slxos-dist` directory.

```
root@VLAB1:~# ls /slxos-dist
extreme-lxc_2.0.0.deb  slxos17r.2.00  slxos17r.2.00.tar.gz  vslx_2.0.0.deb
vslx2.0.0.tar.gz
```



Note

You can also list included directories.

b. Verify that the container's Ethernet interface is up and that an internet address is assigned.

```
root@VLAB1:~# ifconfig -a
eth0      Link encap:Ethernet  HWaddr 00:16:3e:b5:7d:b9
          inet addr:192.0.2.0  Bcast:10.0.0.255  Mask:255.255.255.0
          inet6 addr: 2001:db8::/64 Scope:Link
          inet6 addr: 2001:db8:ffff:ffff:ffff:ffff:ffff:ffff/64 Scope:Global
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:59 errors:0 dropped:0 overruns:0 frame:0
          TX packets:10 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:9877 (9.8 KB)  TX bytes:1312 (1.3 KB)

(output
truncated)
```

c. Enter the **ping** command—specifying a known URL—to verify that you have internet connectivity.

```
root@VLAB1:~# ping example.com
```

6. Install the vSLX debian package.

```
root@VLAB1:~# dpkg -i /slxos-dist/vslx_2.0.0.deb
Selecting previously unselected package vslx-utils.
(Reading database ... 13031 files and directories currently installed.)
Preparing to unpack ./vslx_2.0.0.deb ...
Unpacking vslx (2.0.0) ...
Setting up vslx (2.0.0) ...
```

7. Run the post-installation script.

```
root@VLAB1:~# /VM/postinst-setup.sh
Get:1 http://security.ubuntu.com/ubuntu xenial-security InRelease [102 kB]
Hit:2 http://archive.ubuntu.com/ubuntu xenial InRelease
Get:3 http://archive.ubuntu.com/ubuntu xenial-updates InRelease [102 kB]
Fetched 204 kB in 1s (202 kB/s)
Reading package lists... Done
Reading package lists... Done
```

After successful installation, proceed to [Up-and-running example](#) on page 29.



Appendix C: vSLX on VMware ESXi

[vSLX on VMware overview](#) on page 77

[Resource Scoping for vSLX on VMware](#) on page 78

[Create a VM and install Ubuntu](#) on page 79

[Installation of vSLX on a VM \(linked table\)](#) on page 80

vSLX on VMware overview

You can install vSLX on VMware ESXi virtual machines (VMs).



Note

We assume that you have access to a VMware (ESXi) 6.5 installation and know how to use it.

VMs—as described in this section—support both types of vSLX implementation:

- Host installation: All users share one virtual lab.
- Installation in Linux containers: Multiple users have independent virtual labs.

Implementation flow

The following linked table will guide you in implementing vSLX on VMware:

Table 11: vSLX on VMware implementation flow (host or container)

Task	Topics
Estimate CPU, memory, and storage needs.	Resource Scoping for vSLX on VMware on page 78
Create a VM and install Ubuntu Linux.	Create a VM and install Ubuntu on page 79
Download vSLX and SLX-OS software.	Download vSLX software on page 16
Create an IP-address bridge.	Perform one of the following tasks: <ul style="list-style-type: none">• Configuring a static IP address bridge on page 18• Configuring a dynamic IP address bridge on page 19

Table 11: vSLX on VMware implementation flow (host or container) (continued)

Task	Topics
Copy downloaded vSLX and SLX-OS software.	Copying the software distribution on page 20
Install vSLX.	Perform one of the following tasks: <ul style="list-style-type: none"> • Installing a vSLX lab on a host on page 21 • Container installation (multiple vSLX labs) on page 22

Resource Scoping for vSLX on VMware

Towards installing vSLX on a VMware ESXi virtual machine (VM), prepare a resource-scope table.

The following topics contain scoping information:

- [vSLX server requirements](#) on page 15
- [Scaling Considerations](#) on page 28

Based on the above topics and your implementation scope, estimate your CPU, memory, and storage needs. Towards installing vSLX on a VM, prepare a resource-scoping table, using the following template:

Table 12: vSLX resource-scoping template

CPUs	Memory required	Storage required

vSLX-on-VM scope for OVF export

After you create a vSLX-on-VM instance and set up a virtual lab, you might want to export it as an OVF. You could then import it into another VM, efficiently creating one or more additional virtual labs. Multiple vSLX instances in multiple VMs is one way to configure [Linking across vSLX instances with tunnels](#) on page 37.

If you envision such a scenario, at what level of configuration should you export the OVF? Some considerations:

- Export a fully configured instance, which might save configuration time after importing the OVF.
- Export a "lean and mean" instance—for example, one SLX 9540 *chassis* to another SLX 9540 *chassis*—which might be easier to copy over your network.

Create a VM and install Ubuntu

These tasks create a VMware virtual machine configured to support vSLX and then install Ubuntu Linux on the VM.



Note

The instructions in these tasks are for VMware vCenter agent. Some details may vary for vSphere.

Initializing a VM for vSLX

As you begin creating a VMware VM, specify a name, VM-compatibility, Linux version, and datastore.

1. Log in to VMware ESXi.
2. In the left-hand **Navigator** frame, select **Virtual Machines**, and then click **Create/Register VM**.
3. On the **Select creation type** screen, select **Create a new virtual machine**, and then click **Next**.
4. On the **Select a name and guest OS** screen:
 - a. In the **Name** field, type a descriptive and unique VM name.
 - b. In the **Compatibility** list, select **ESXi 6.5 virtual machine**.
 - c. In the **Guest OS family** list, select **Linux**.
 - d. In the **Guest OS version** list, select **Ubuntu Linux(64-bit)**.
 - e. Click **Next**.
5. On the **Select storage** screen, select the destination datastore, and then click **Next**.

Next: [Specifying VM settings for vSLX](#) on page 79

Specifying VM settings for vSLX

Specify CPU, memory, and storage scope; SCSI controller, network adapter, and datastore settings.

1. On the **Virtual Hardware** tab of the VMware ESXi **New virtual machine > Customize settings** screen, expand the **CPU** section and specify the following settings:
 - a. For the **CPU** and **Cores per socket** values, refer to your [Resource Scoping for vSLX on VMware](#) on page 78 calculations.
 - b. Select **Expose hardware assisted virtualization to the guest OS** and **Enable virtualized CPU performance counters**.



Note

The other **CPU** fields are "don't care".

2. For the **Memory**, and **Hard disk 1** values, refer to your [Resource Scoping for vSLX on VMware](#) on page 78 calculations.
3. Make sure that the **SCSI Controller 0** value is **LSI Logic Parallel**.



Note

The **SATA** and **USB** fields are "don't care".

4. Expand the **Network Adapter 1** section and specify the following settings:
 - a. Make sure that the **Network Adapter 1** value is **VM Backplane**.
 - b. Make sure that the **Adapter Type** value is **VMXNET 3**.
 - c. Make sure that **Connect at power on** is selected.
5. Expand the **CD/DVD Drive 1** section and specify the following settings:
 - a. Make sure that **Connect at power on** is selected.
 - b. Make sure that the **CD/DVD Drive 1** value is **Datastore ISO file**.
 - c. In the **Datastore browser**, select **ubuntu-16.04-x-server-amd64.iso** and then click **Select**.
 - d. On the **Customize settings** screen, click **Next**.
6. On the **Ready to complete** screen, examine your settings, and make needed corrections.

Next: [Installing Ubuntu Linux on the VM](#) on page 80

Installing Ubuntu Linux on the VM

Conclude VM creation and install Ubuntu Linux on the VM.

1. On the **Ready to complete** screen, examine your settings, and then click **Finish**.
2. On the **Virtual Machines** screen, select the VM that you created and then click **Power on**.
3. Click the Console and then configure your language settings.
4. Click **Install Ubuntu Server**.

Next: [Installation of vSLX on a VM \(linked table\)](#) on page 80

Installation of vSLX on a VM (linked table)

After you create a VMware VM and install Ubuntu Linux on it, the following linked table guides you through installing vSLX on the VM.

Table 13: Installing vSLX on a VM (host or container)

Task	Topics
Download vSLX and SLX-OS software.	Download vSLX software on page 16
Create an IP-address bridge.	Perform one of the following tasks: <ul style="list-style-type: none"> • Configuring a static IP address bridge on page 18 • Configuring a dynamic IP address bridge on page 19
Copy downloaded vSLX and SLX-OS software.	Copying the software distribution on page 20
Install vSLX.	Perform one of the following tasks: <ul style="list-style-type: none"> • Installing a vSLX lab on a host on page 21 • Container installation (multiple vSLX labs) on page 22